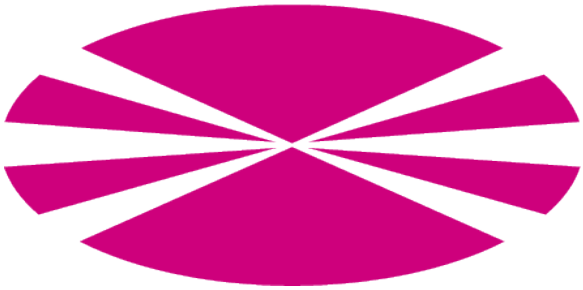


FACULTADE DE INFORMÁTICA

UNIVERSIDADE DA CORUÑA

Proyecto de Fin de Carrera de Ingeniería Informática

Simulación de elementos elásticos mediante FFD con volúmenes de Bézier

Autor: Gabriel Sanmartín Díaz

Directores: Julián Flores González

José Carlos Dafonte Vázquez

A Coruña, Julio de 2010

Especificación

Título: Simulación de elementos elásticos mediante FFD con volúmenes de Bézier

Clase: Proyecto de investigación y desarrollo

Autor: Gabriel Sanmartín Díaz

Directores: Julián Flores González
José Carlos Dafonte Vázquez

Tribunal:

Fecha de lectura:

Cualificación:

Agradecimientos

A mi familia, a Iria y a mis amigos por su apoyo, y especialmente a Julián Flores y Carlos Dafonte por su paciencia y su ayuda en el desarrollo de este proyecto.

Resumen

En este proyecto se presenta un estudio de varias técnicas de deformación en tiempo real de objetos en entornos 3D, destinadas al modelado de animaciones y comportamientos de personajes en mundos virtuales.

Se ha creado, partiendo de una API gráfica convencional y a nivel de código (OpenGL) un pequeño motor gráfico para la representación, deformación y animación de personajes en tiempo real; pudiendo emplearse con posterioridad para desarrollos más ambiciosos (videojuegos, animaciones, aplicaciones de realidad virtual…). Entre las capacidades proporcionadas se encuentran la representación jerárquica de objetos (con modelos 3D y texturas), el soporte para sistemas óseos y las capacidades de cinemática inversa.

El objetivo de este motor será proporcionar una infraestructura para el desarrollo de una nueva técnica para dotar de comportamiento elástico a estos personajes, de tal manera que dicho comportamiento, aun teniendo en cuenta su naturaleza imposible, se perciba como intuitivamente correcto de cara al usuario. La creación de esta infraestructura en lugar del uso de algún motor ya existente se debe a la necesidad de tener una flexibilidad imposible de obtener ante la rigidez de las interfaces de motores ya existentes.

Esta deformación elástica deberá poder aplicarse en tiempo real, de tal manera que habrá de hallarse un compromiso entre realismo y eficiencia; ofrecerse posibilidades de escalabilidad y una completa integración con las técnicas ya existentes para la animación de personajes.

Para proporcionar todos estos requisitos la técnica empleada se ha basado en el uso de volúmenes de Bézier para generar las deformaciones sobre los vértices, de forma indirecta, a través de la transformación de su sistema de coordenadas. Esta estrategia, llamada Free-Form Deformation, ha demostrado ser una alternativa flexible, escalable y eficiente.

Palabras Clave

Deformación, vértices, computación gráfica, 3D, OpenGL, deformación ósea, Free-Form Deformation, Cinemática Inversa, Mass-Spring System

Índice general

Índice de figuras

Índice de tablas

1. Introducción

1.1. Motivación

En los últimos años la animación generada por ordenador ha sido una de las ramas de la computación gráfica que más se ha desarrollado, impulsada por el cine de animación y los videojuegos, creándose cada vez entornos con mayor realismo, movimientos más fluidos y mayores exigencias en cuanto a cómputo.

En estos dos ámbitos las ampliaciones son múltiples, apareciendo nuevos "avatares[1]" con movimientos y deformaciones que resultan imposibles en la vida real pero sí tienen sentido dentro del mundo descrito por el entorno animado. Las reglas físicas en este tipo de escenarios pueden ser muy distintas a la realidad, pero aun así los resultados deben ser intuitivamente coherentes con el mismo: incluso cuando estos movimientos o deformaciones puedan ser en la práctica imposibles, sí deben resultar visualmente plausibles para el receptor, de manera que éste los perciba como "intuitivamente consistentes". En el caso de los videojuegos aplican reglas similares, y esta idea de "consistencia intuitiva" se mantiene.

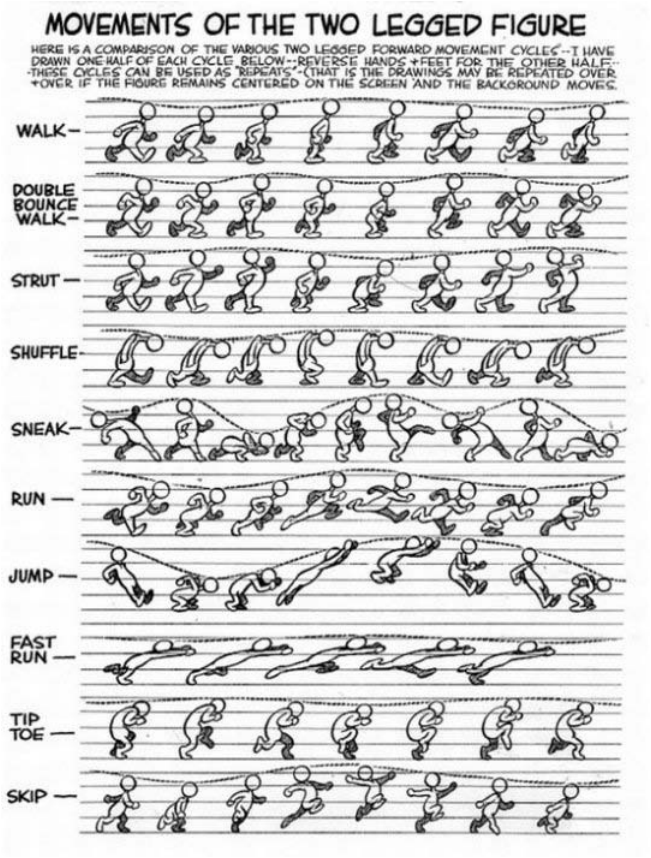


Figura 1: Ejemplo de animación tipo Cartoon

Un ejemplo de ello son los entornos de estética "*cartoon*" o dibujos animados, en los cuales los personajes, dibujados ya desde una base de tipo caricaturesco, desarrollan movimientos muy

[1] representación del usuario o su alter-ego en forma de modelo tridimensional utilizado en los videojuegos y otros sistemas virtuales

exagerados que en absoluto se corresponden con la realidad pero si resultan intuitivamente consistentes al seguir ciertas reglas físicas: un personaje se contrae, aunque exageradamente, antes de estirarse para saltar y volver a contraerse al caer de nuevo [27].

El caso de estudio plantea la existencia de un personaje "elástico", cuyos miembros podrán estirarse o contraerse de tal manera que al liberarse la fuerza que los deforma se vuelva a una situación inicial de reposo siguiendo un movimiento similar al de una goma elástica en tales condiciones.

Este planteamiento implica la deformación del objeto 3D a nivel de vértice, de tal modo que es necesario la accesibilidad directa a dichos vértices. Es debido a estas exigencias de flexibilidad que el modelo propuesto requiere el desarrollo de un paradigma completo de deformación del modelo del personaje; partiendo desde la API que a bajo nivel provee el motor gráfico (OpenGL), y escalando a partir de ahí todas las técnicas de deformación previas a la aplicación del problema planteado (representación de objetos, deformación ósea, animación de personajes…). Esto, a diferencia de utilizar un motor gráfico plenamente desarrollado (en el cual el acceso a vértices es más complejo y el control sobre los resultados viene condicionado por la rigidez de la interfaz de programación) permitirá contar con un control directo desde el más bajo nivel. Sobre esta base se crearán las subsiguientes capas de abstracción que procedan, creándose en última instancia una interfaz sencilla de programación.

Intentaremos, en resumen, crear un paradigma sencillo, intuitivo y computacionalmente eficiente para desarrollar este tipo de movimiento a bajo nivel (vértices) que permita contraer o expandir el conjunto de los mismos de forma computacionalmente manejable, y que además permita su aplicación de forma local. En el caso especial de un personaje animado, será necesario también hacer que la deformación local pueda compatibilizarse con la de un modelo articulado dotado de deformación ósea. Todas estas técnicas de deformación serán desarrolladas de forma integral en el proyecto.

1.2. Acerca de esta memoria

La presente memoria se ha estructurado en base a la metodología de desarrollo escogida (ciclo de vida incremental, ver siguiente capítulo), siguiendo los diferentes incrementos abordados y tratando las diferentes fases de su desarrollo por separado.

En primer lugar se ha realizado una **introducción** que describe el sistema, su contextualización y la motivación del presente trabajo.

A continuación se detalla la elección de la estrategia de ciclo de vida o **metodología de trabajo** empleada, justificando su adecuación a las características del presente proyecto y el plan de trabajo para su puesta en práctica.

En tercer lugar se exponen los aspectos relativos a la **planificación** abordada para el proyecto en los momentos previos al inicio de su ejecución, como son la gestión de tiempos, desarrollo del cronograma y la estructura de desglose de trabajo o la gestión y previsión de costes.

La sección siguiente aborda una descripción general de **conceptos y tecnologías utilizadas**, como base técnica para la comprensión del resto del documento, exponiendo de manera global los fundamentos teóricos sobre lo que se sustenta el proyecto.

Los siguientes cinco capítulos detallan el **desarrollo de las cinco revisiones** planificadas como resultado de la aplicación del ciclo de vida incremental, cada una de las cuales, al desarrollarse secuencialmente y en cascada, contemplará a su vez las respectivas fases de Análisis, Diseño e Implementación, aspectos que se abordarán en detalle para los incrementos en los que sea relevante, tras una descripción inicial de los objetivos de cada revisión. Por último, al final de cada revisión se detallará el seguimiento del avance técnico del proyecto frente a la línea de base, efectuado a la consecución del hito de fin de fase.

Por último se aborda una exposición de las **conclusiones y lecciones aprendidas**, las aportaciones del modelo desarrollado y las vías de trabajo futuro que por falta de tiempo o recursos resultó imposible ejecutar en el presente proyecto.

2. Metodología de trabajo: ciclo de vida incremental

Para el desarrollo del proyecto se ha escogido, por considerarse más apropiado, un modelo evolutivo de carácter iterativo, como es el caso del modelo de ciclo de vida incremental [43]. Este modelo se basa en la filosofía de construir gradualmente las funcionalidades o capacidades del programa en cada revisión, a través del desarrollo en módulos.

Básicamente el ciclo de vida incremental trabaja como una repetición continua del ciclo de vida en cascada, en el que cada incremento trabaja según este modelo de ciclo de vida, con sus fases secuenciales de análisis, diseño, codificación y pruebas, y en el que la salida de cada incremento supone una entrega con funcionalidades añadidas sobre la versión anterior. Es decir, un programa funcional (no un prototipo, con aspectos que no funcionan o lo hacen de manera incompleta) pero con capacidades gradualmente mayores a medida que se avanza en el desarrollo.

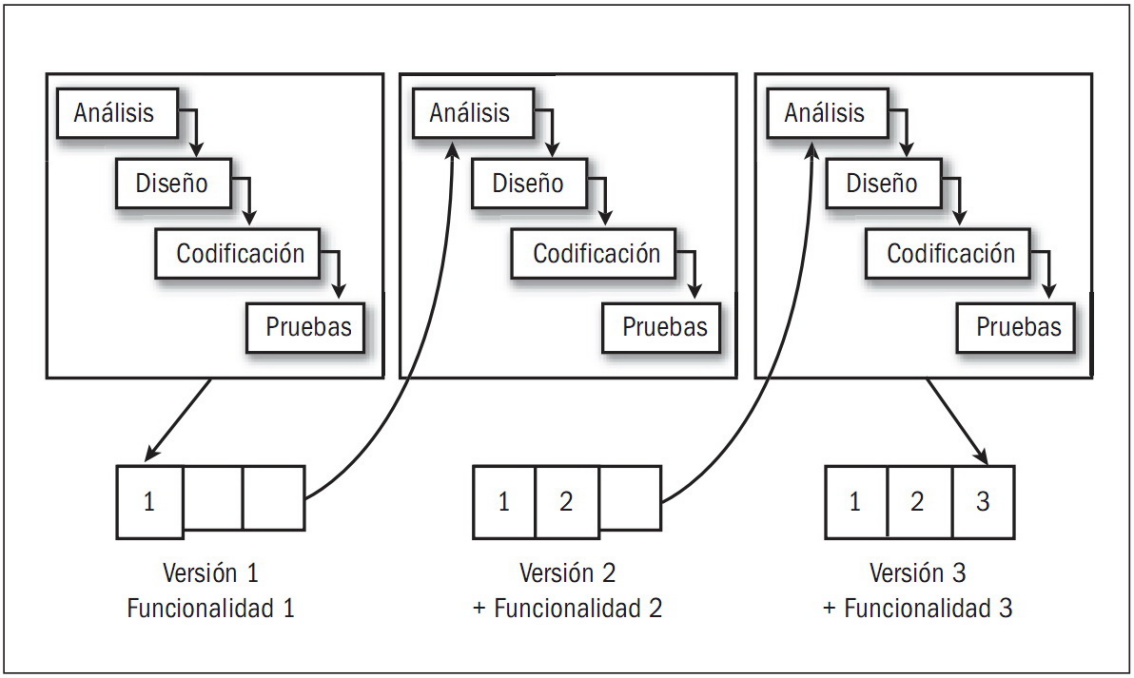


Figura 2: Ciclo de vida incremental

Al tratarse de un proyecto de investigación, los riesgos asociados son elevados y existe una gran incertidumbre en el proceso de desarrollo, derivada de elecciones que pueden convertirse en incorrectas al alcanzar la iteración siguiente en el ciclo de vida.

El ciclo de vida incremental, aun cuando se concibe inicialmente para sistemas comerciales de gran envergadura que se desarrollan en versiones de funcionalidad limitada para ir amortiguando costes a medida que se avanza en el desarrollo, presenta beneficios que sí pueden entenderse como idóneos para el proyecto desarrollado, tales como:

-El riesgo de un proyecto grande se reduce al trabajar sobre proyectos pequeños.

-Al desarrollarse las funcionalidades gradualmente y en incrementos, es más fácil determinar si los requisitos analizados para los siguientes subniveles son correctos. Esto resulta especialmente útil para proyectos de investigación en los que la incertidumbre es mayor y los requisitos suelen ser ambiguos.

-No es necesario contar con una especificación completa de los requisitos de fases avanzadas para comenzar a trabajar; se parte de una idea genérica (ej. “creación de personajes elásticos”), que se puede ir depurando en sus requisitos en estados más avanzados del desarrollo, sin detenerse en el mismo mientras se desarrollan funcionalidades que se sabe van a ser necesarias (p. ej. Carga de modelos 3D).

-En el caso de detectarse un error en el diseño o análisis, resulta más sencillo “volver atrás” en el desarrollo del proyecto, simplemente deshaciendo una iteración anterior. Esto resulta especialmente indicado para proyectos de investigación, en el que la aplicación de un algoritmo en un punto determinado puede relevarse como inadecuada más adelante (p. ej. En el caso de la cinemática inversa existen varios algoritmos para resolver esta clase de problemas; algunos más indicados para ciertas exigencias que otros).

Se entiende el proceso de desarrollo como dentro del paradigma incremental y no como un proyecto basado simplemente en el modelo en cascada en el que se trabaja “funcionalidad a funcionalidad”, porque cada una de las versiones planteadas es un programa **funcional** que, aunque incompleto para lo exigido en los requisitos del proyecto, sí permite realizar tareas al 100% de funcionalidad (a diferencia de un prototipo); además los procesos secuenciales de análisis, diseño, codificación y pruebas característicos del ciclo de vida en cascada, se desarrollan específicamente para cada incremento.

Otros beneficios del ciclo de vida incremental quizá menos ajustados para este caso por su naturaleza (un único miembro en el equipo, no existe cliente como figura interesada del proyecto) serían:

-Al reducirse el tiempo de desarrollo del sistema (cada uno de los incrementos) decrecen las probabilidades de que el cliente modifique los requisitos durante el desarrollo.

-El ciclo de vida incremental facilita el trabajo en paralelo por parte de varios equipos, que pueden trabajar sobre incrementos diferentes acordando una interfaz común previamente entre ambas revisiones.

3. Planificación del proyecto

3.1. Planificación y gestión de tiempos

Se ha planificado el desarrollo del proyecto en base a la compatibilización del mismo por parte del alumno con las horas desempeñadas en su puesto de trabajo (a tiempo parcial), con lo que se estima un máximo de 3 horas de trabajo diarias durante la práctica totalidad del curso 2009-2010 (septiembre - junio), cinco días a la semana, lo que da un total de 621 horas.

Al inicio del desarrollo se tienen en cuenta las fases de inicio y definición de proyecto, así como el estudio de viabilidad y alcance del mismo en líneas generales, antes de abordar por separado (y como resultado de la elección del ciclo de vida incremental) el análisis, definición, diseño y desarrollo de cada iteración por separado, teniendo siempre en cuenta la idea global del proyecto.

El desarrollo en sí se ha llevado a cabo en base a los hitos que representan las diferentes entregas de las versiones planificadas en el proyecto, hasta la entrega de la versión 5 con la integración de todas las técnicas utilizadas y la finalización del cumplimiento de los requisitos planteados. Para cada una de estas iteraciones se han planificado sus respectivas fases dentro del ciclo de vida en cascada, esto es, estudio de alcance, análisis, diseño, implementación, pruebas. La secuencialidad en el desempeño de todas estas tareas es absoluta debido a la limitación impuesta por la existencia de un único recurso (el alumno firmante), salvo en el caso de la documentación, para la cual se ha ido dedicando tiempo a medida que el desarrollo se llevaba a cabo, entendiendo que este proceso de documentación forma parte integral e indivisible de cualquier desarrollo.

A posteriori se ha tomando en consideración la planificación de la elaboración de esta memoria y la presentación asociadas al trabajo de proyecto de fin de carrera.

Según la planificación realizada el número de horas de trabajo asciende a 621 horas, desempeñadas en los días laborables del 7 de septiembre de 2009 hasta la finalización de esta memoria, el día 16 de junio de 2010.

Cada uno de los hitos representados por la entrega de las versiones resultado de los entregables de cada iteración en el ciclo incremental representan puntos de seguimiento por parte de los directores del proyecto, evaluación del cumplimiento de los requisitos y luz verde para el inicio de la siguiente revisión con los cambios pertinentes si los hubiere y con los nuevos requisitos construidos en base a lo desarrollado. Esto supone la celebración de una reunión con uno, otro o ambos directores a la consecución de cada hito de revisión.

Entregables e hitos del proyecto

En el desarrollo del proyecto se establecen los siguientes hitos para marcar el avance técnico del mismo y representar la consecución de los entregables en el momento de alcanzar el hito. El establecimiento de estos hitos se ha realizado en base a la finalización de los diferentes incrementos según la metodología de ciclo de vida escogida (incremental).

ID	Hito	Entregable	Descripción
H00	***Inicio del proyecto***	Propuesta de proyecto de fin de carrera	Documento que detalla una propuesta de proyecto de fin de carrera para su aprobación por los directores del proyecto, abarcando su alcance y una definición preliminar de requisitos.
H01	***Versión 1 operativa***	Revisión base, estructura del marco 3D	Constituye la base inicial de trabajo sobre la que se construyen el resto de revisiones. Por sí misma, representa una aplicación para la carga y representación de objetos y escenas 3D, con soporte para texturas y jerarquías de objetos.
H02	***Versión 2 operativa***	Revisión con Deformación ósea	Sobre la revisión anterior se agrega el soporte integral para la definición de personajes con esqueletos de deformación de forma integral, pudiendo animarse dichos personajes o avatares mediante la transformación de sus huesos.
H03	***Versión 3 operativa***	Revisión con soporte para cinemática inversa	Permite alcanzar soluciones de cinemática inversa en tiempo real sobre la estructura de deformación ósea creada en la revisión anterior. La revisión supone la posibilidad de crear animaciones sobre la marcha.
H04	***Versión 4 operativa***	Revisión con soporte para la creación de elementos elásticos	Permite la creación, flexible y escalable, de elementos elásticos individuales mediante la creación de deformadores. Esta revisión trabaja por separado respecto de las deformaciones anteriores.
H05	***Versión 5 operativa***	Revisión final integrada y optimizada	Integra la deformación elástica con la deformación ósea y la cinemática inversa, así como otros elementos desacoplados. Optimización general del sistema.
H05	***Cierre de proyecto***	Aplicación terminada, Memoria de Proyecto.	Proporciona la versión definitiva de la aplicación acompañada de esta memoria.

Tabla 1: Hitos y entregables del proyecto

Esta tabla representa, en visión general, el conjunto de entregables constituyentes del proyecto, pudiendo ir más abajo en el nivel interno de desarrollo y contemplar los diferentes entregables dentro de cada unidad de trabajo (documentos de análisis, diseño, prototipos intermedios, revisiones parciales implementadas, documentos de resultado de pruebas, etc.).

La fecha de alcance de los hitos previstos, según la planificación realizada, puede consultarse en la figura 3.

Nombre de tarea	Duración	Comienzo
Desarrollo proyecto fin de carrera	**207 días**	**mar 01/09/09**
Inicio de proyecto	0 días	mar 01/09/09
Versión 1 operativa	0 días	jue 05/11/09
Versión 2 operativa	0 días	mar 29/12/09
Versión 3 operativa	0 días	vie 26/02/10
Versión 4 operativa	0 días	mar 11/05/10
Versión 5 operativa	0 días	mié 09/06/10
Cierre de proyecto	0 días	mié 16/06/10

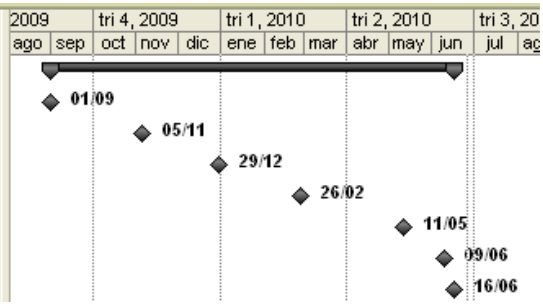


Figura 3: Fechas previstas de alcance de hitos

Diagrama de Gantt

A continuación se presenta el diagrama de Gantt completo asociado al proyecto y establecido como línea de base del mismo.

Fecha de inicio del proyecto:	Lunes 7 de septiembre del 2010
Fecha de finalización prevista:	Miércoles 16 de junio del 2010
Duración:	207 días (7 meses), 621 horas

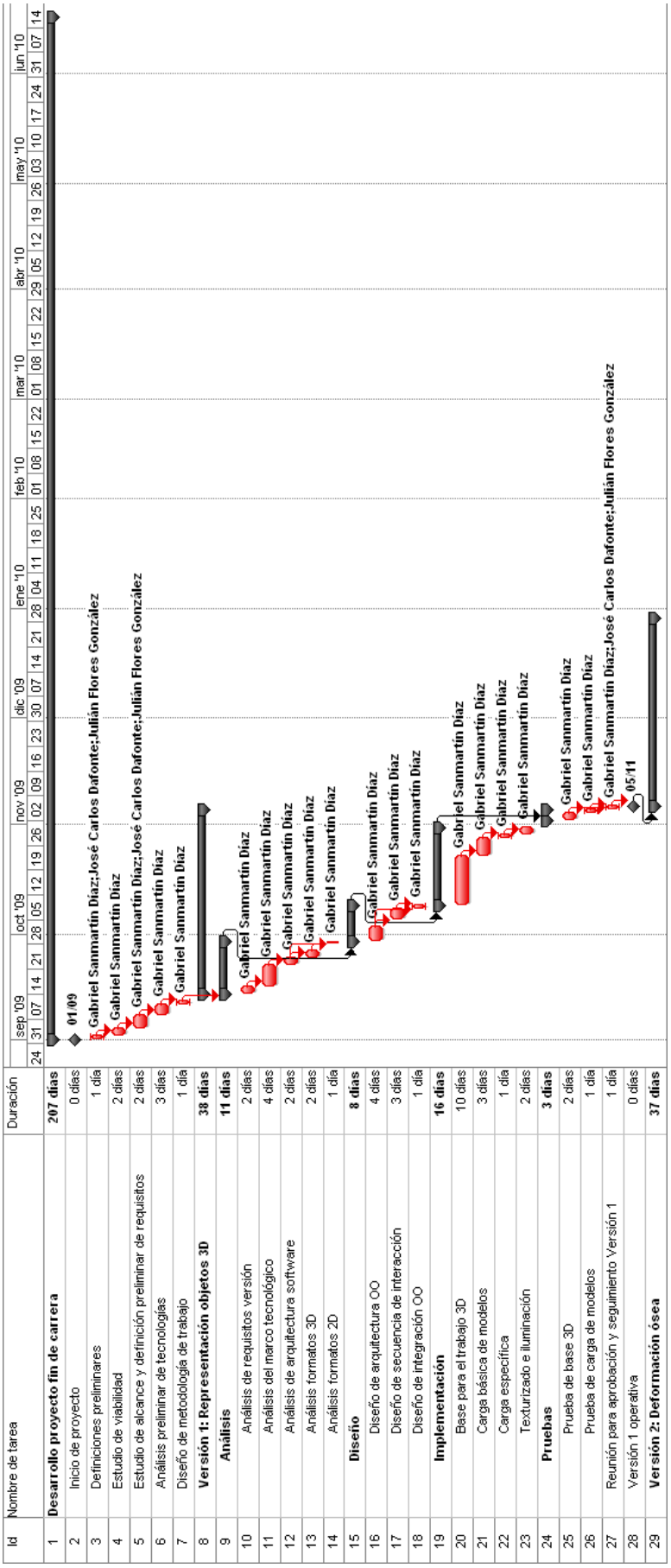


Figura 4: Diagrama de Gantt del proyecto (i)

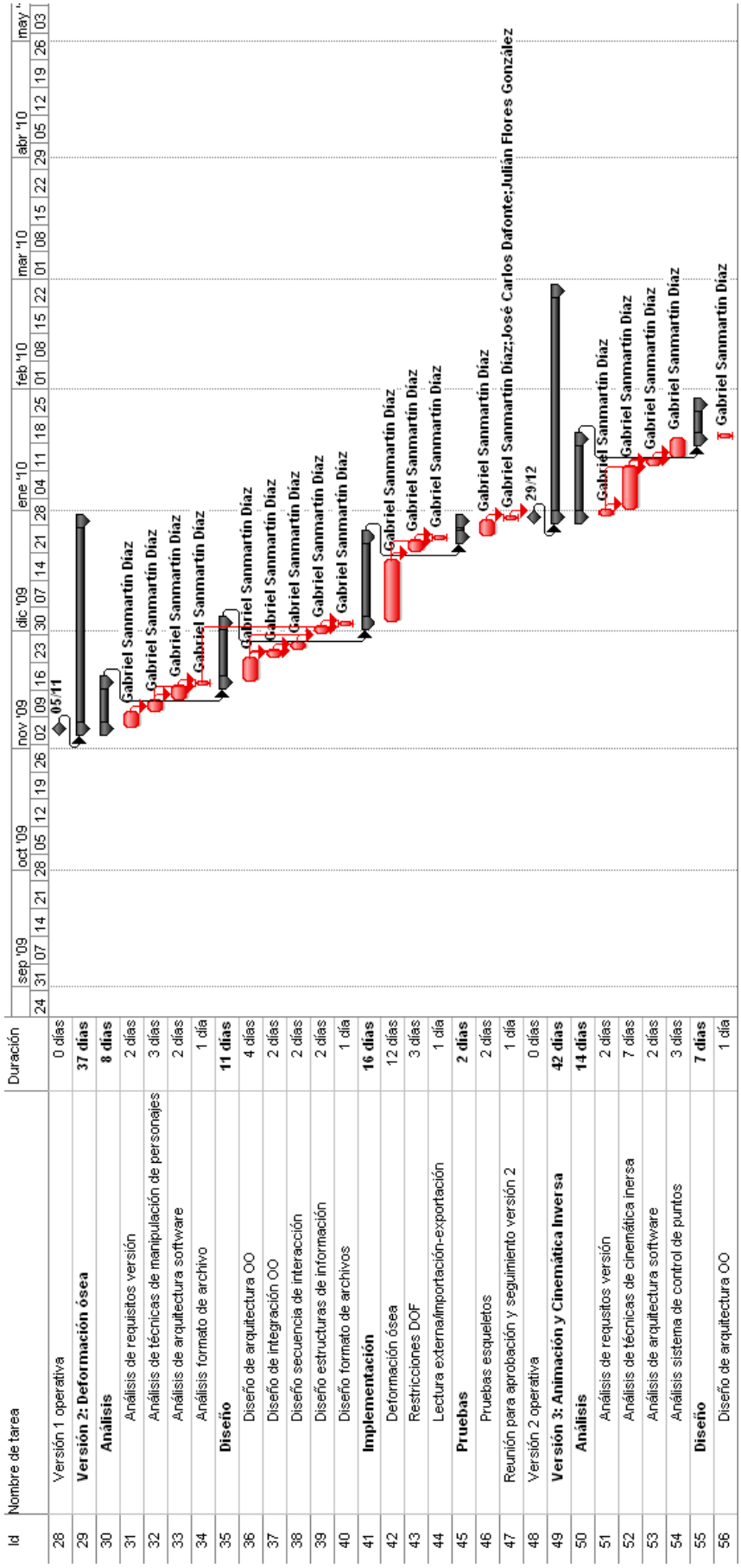


Figura 5: Diagrama de Gantt del proyecto (ii)

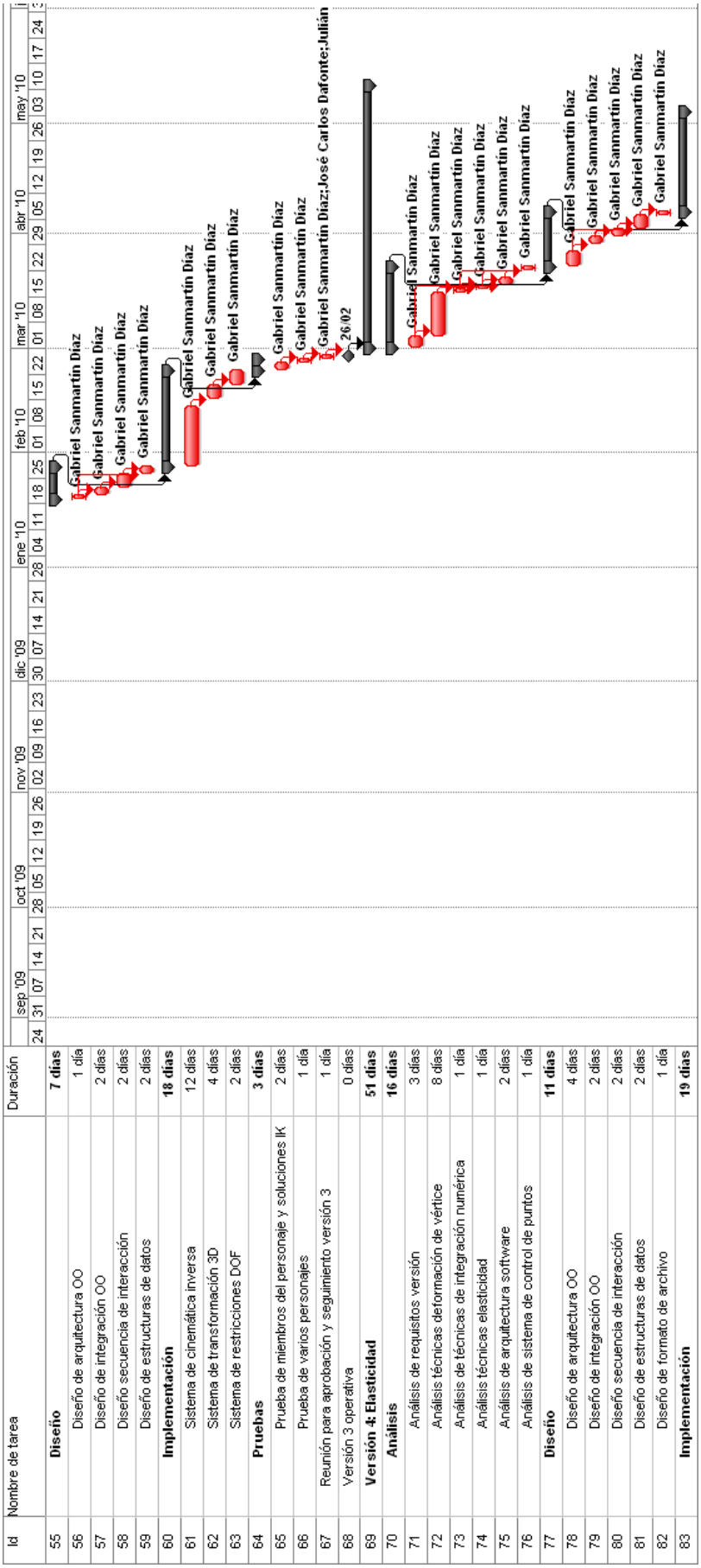


Figura 6: Diagrama de Gantt del proyecto (iii)

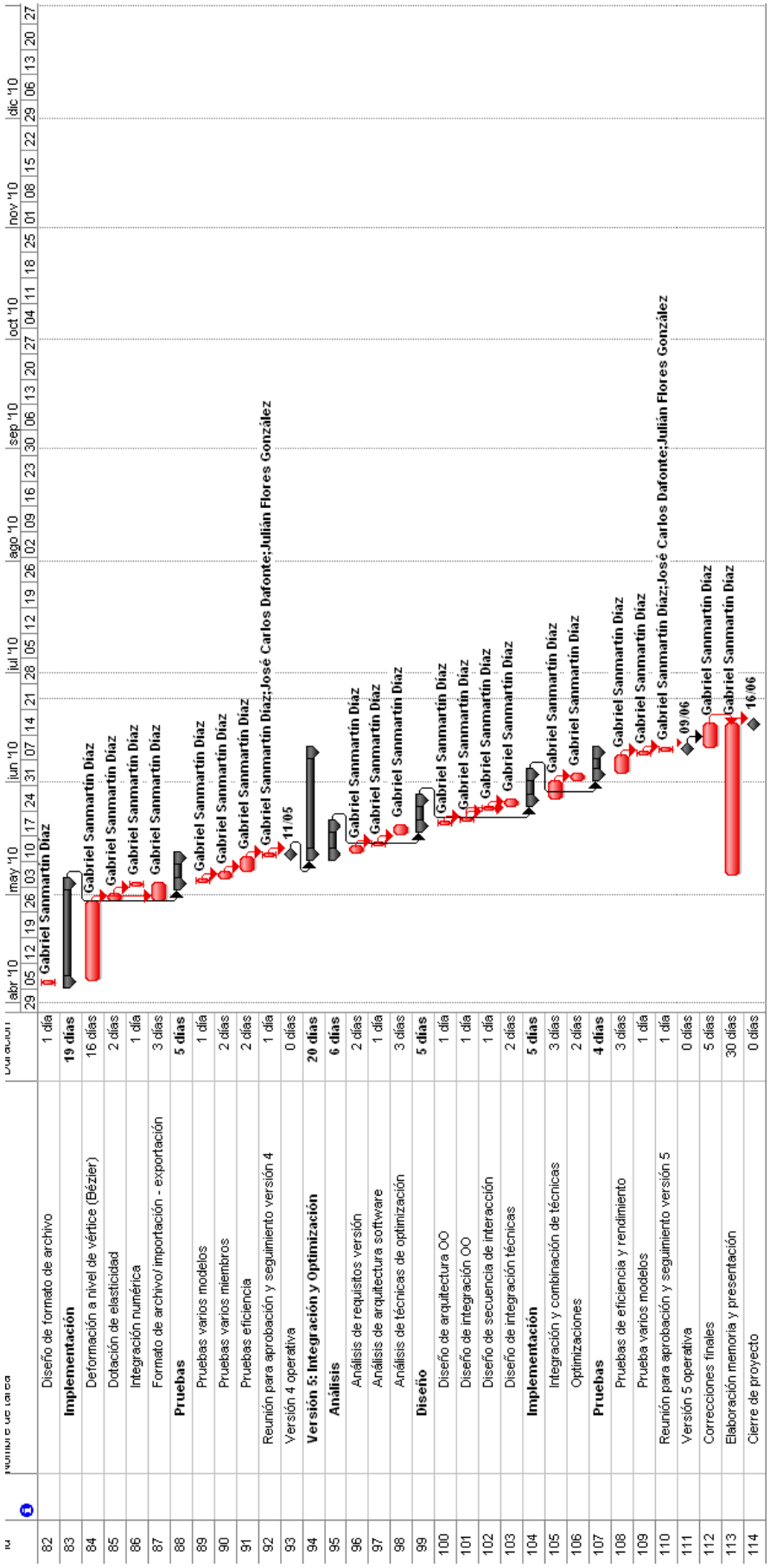


Figura 7: Diagrama de Gantt del proyecto (iv)

3.2. Planificación y gestión de costes

Los costes del proyecto se han estimado como resultado de la suma de los costes materiales necesarios y humanos (los recursos humanos dedicados exclusivamente al proyecto durante el tiempo de desarrollo).

Recursos humanos

El proyecto se plantea como un desarrollo de algo menos de 7 meses con un trabajo diario de 3 horas, para su compaginación con el desempeño del puesto de trabajo del alumno. En base al equipo construido, se estima la dedicación de cada recurso en el desarrollo integral del proyecto y en función de esta dedicación se han calculado los costes asociados al trabajo del mismo.

Una vez realizado el desglose de las tareas previstas se estimó como necesarias una persona a tiempo completo (un programador), más la supervisión de los directores de proyecto en los momentos puntuales de reuniones y especificación conjunta de requisitos.

Función	Nombre	Dedicación (horas)	Coste unitario	Total
Desarrollador	Gabriel Sanmartín	Completa (621 horas)	25	15525
Director de proyecto	Julián Flores	Completa (60 horas)	45	2700
Director de proyecto	Carlos Dafonte	Completa (60 horas)	45	2700
			TOTAL:	20925€

Tabla 2: Desglos de costes de recursos humanos

Recursos materiales

Los recursos materiales derivan únicamente de las necesidades hardware para poner en práctica los requisitos software inherentes al proyecto. Al tratarse de un desarrollo cuyo objeto es la ejecución en cualquier ordenador personal de propósito general, los requisitos hardware se limitan a una estación de trabajo preparada para el cómputo intensivo con tareas 3D; o lo que es lo mismo, se requiere la presencia de una tarjeta 3D con soporte para el estándar OpenGL 2.0, lo que hoy en día supone la práctica totalidad de ordenadores personales.

Sin embargo, para la correcta ejecución a nivel de rendimiento del sistema, se requiere al menos de un procesador Pentium 4 a 3Ghz con una aceleradora igual o superior a la generación FX 52xx de nVidia (una tarjeta que, de todos modos, fue lanzada al mercado hace más de 5 años, por lo que cualquier tarjeta disponible actualmente valdría para la ejecución correcta del programa).

En cuanto a las plataformas software utilizadas, como por ejemplo el backend gráfico 3D o la API de manejo de ventanas, no comportan gasto alguno al tratarse de software de código abierto y por tanto de uso gratuito. No se reportan por tanto gastos asociados a la infraestructura software adquirida.

La elección de software libre para el backend gráfico y el sistema de ventanas viene condicionada por la idea de ofrecer la mejor oferta posible sin perder calidad en el desarrollo,

y en este sentido se ha considerado que OpenGL, ofrece sobradamente las capacidades técnicas y funcionalidades requeridas para las características de este proyecto; máxime cuando la filosofía de código abierto va muy en consonancia con la idiosincrasia investigativa del desarrollo.

Nombre material	Cantidad	Coste unitario	Total
Ordenadores	1	1200/ud.	1200
		TOTAL:	1200

Tabla 3: Desglose de costes materiales

Presupuesto final

Total Recursos Humanos	59500
Total Hardware	1200
TOTAL:	68658

Tabla 4: Presupuesto final

4. Tecnologías utilizadas y fundamentos teóricos

4.1. OpenGL

Open Graphics Library (OpenGL) es un conjunto de librerías que definen un interfaz software entre las aplicaciones y el hardware gráfico. La librería está formada por unas 150 instrucciones diferentes que se utilizan para especificar los objetos y las operaciones necesarias para desarrollar aplicaciones interactivas tridimensionales.

OpenGL ofrece independencia con respecto a la plataforma de hardware y el sistema operativo en que se trabaje, brindando con ello una enorme portabilidad a sus productos. Hoy en día es, junto con las librerías DirectX de Microsoft (sólo disponibles para sus sistemas), el soporte base gráfico más empleado.

Así, OpenGL permite, entre otras cosas:

- Construir formas geométricas a partir de primitivas
- Ubicar los objetos en el espacio tridimensional y seleccionar el punto de vista de la escena
- Aplicar el color a los objetos, ya sea mediante una asignación explícita de la aplicación, a partir de las condiciones de iluminación o mediante la utilización de texturas.
- Convertir la descripción matemática de los objetos y la información sobre el color en píxeles de la pantalla, proceso que se llama rasterización.

Aunque OpenGL está concebido para diseñar aplicaciones interactivas y facilita al usuario herramientas como la selección, sus capacidades resultan insuficientes para, entre otras cosas, crear interfaces gráficas con un mayor grado de interactividad.

Estas limitaciones condujeron al desarrollo de librerías como AUX, GLUT o GLUI.

Glut

Glut es un interfaz de programación con C ANSI o C++ que proporciona una API de programación de aplicaciones basadas en ventanas para simplificar el aprendizaje y creación de programas en OpenGL independientes del sistema operativo.

Las librerías GLUT ofrecen, entre otras cosas, las siguientes prestaciones:

- Ventanas múltiples para render
- Procesamiento de eventos de entrada iniciados por el usuario (callbacks)
- Variados dispositivos de entrada
- Menús desplegables
- Rutinas para generar objetos estándar
- Etc.

Glui

GLUI es una librería basada a su vez en GLUT para simplificar aun más el desarrollo de interfaces gráficas de usuario (GUI). Proporciona de forma sencilla e independiente de la plataforma controles como botones, casillas de verificación, botones radiales o selectores para su uso en aplicaciones OpenGL. Es independiente del sistema de ventanas, delegando en GLUT la gestión de los problemas dependientes de la plataforma, como la gestión de ventanas o el ratón. Algunas de sus características son:

- Creación de ventanas con una línea de código
- Soporte para múltiples ventanas
- Soporte para controles de usuario como:
- Botones
- *Checkboxes* o casillas de verificación
- Botones radiales para opciones mutuamente exclusivas
- Cajas de texto editables para cadenas de texto, enteros o valores de punto flotante
- "*Spinners*" para enteros y valores de punto flotante
- Campos de texto estático
- Paneles para agrupar conjuntos de controles
- Líneas de separación para organizar los controles
- Generación de callbacks ante un cambio de valor
- Variables "live"
- Manejo de los controles a través de teclado.

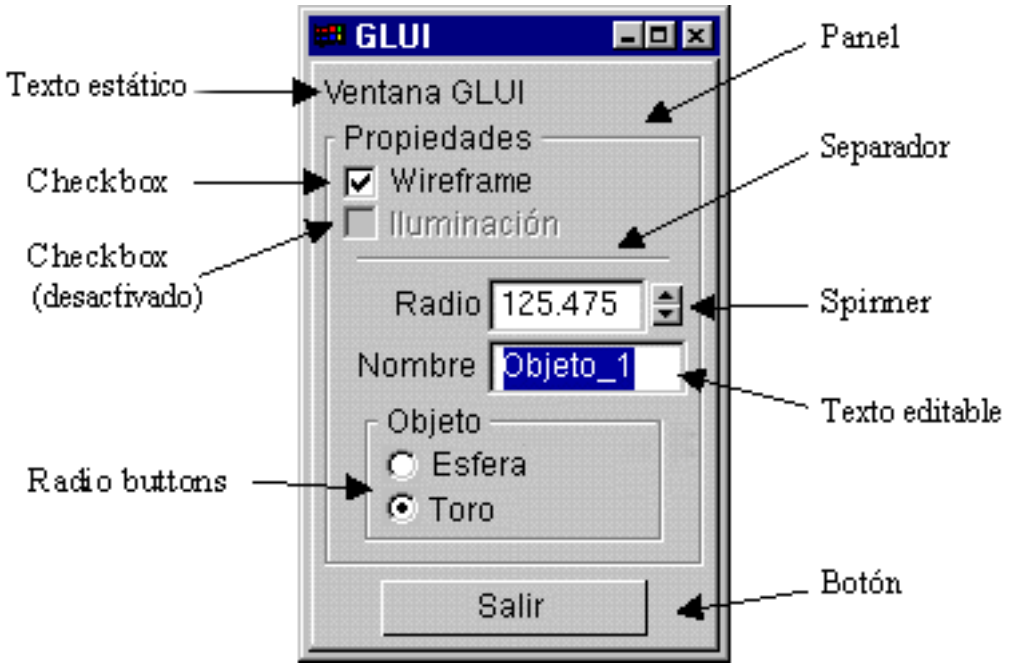
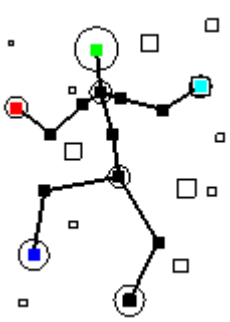


Figura 8: Ejemplo de ventana GLUI

4.2. Cinemática Directa vs. Cinemática Inversa

A la hora de estudiar el movimiento, existen dos aproximaciones diferentes: la cinemática y la cinética. La cinemática es la ciencia del movimiento sin tener en cuenta las fuerzas que lo

originan. Cuando entran en juego las fuerzas y torques sobre un objeto para crear movimiento, hablamos del aspecto cinético de la dinámica.



En el caso de estudio se desea originar movimiento en un personaje pero sin tener en cuenta las fuerzas que lo originan, pues no resulta realmente necesario estudiarlas (ello supondría el estudio de las tensiones de músculos sobre los huesos, algo que no es necesario en el modelo planteado).

La cinemática tiene su fundamento básicamente en la geometría del movimiento. Habitualmente, para animar un personaje se construye una jerarquía ósea que representa las diferentes partes del mismo. Al aplicarse una animación, lo que se está haciendo es registrar la posición y orientación de cada una de estas partes. Por ejemplo, para mover la mano hasta una posición deseada, rotaría el antebrazo, luego el brazo y finalmente la mano, hasta llegar al objetivo. Ésta técnica de animación se conoce como **cinemática directa (*Forward Kinematics*, FK)**.

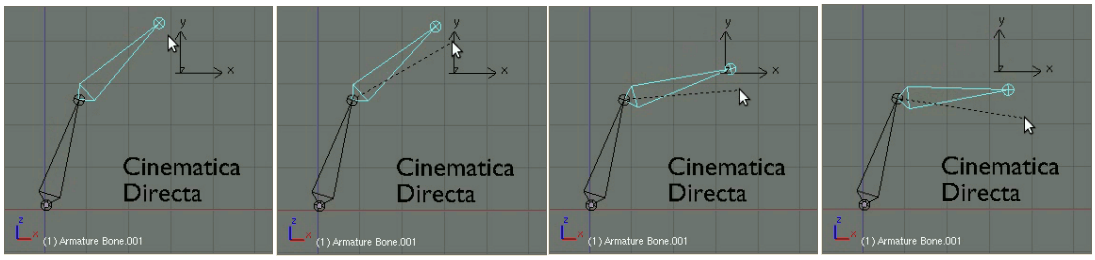


Figura 9: Cinemática Directa

Si por el contrario interesa más simplemente hacer saber al programa dónde se desea posicionar el elemento de interés o efector (la mano, por ejemplo) y que sea éste quien se encargue de calcular la orientación y posición del resto de huesos para alcanzar dicha posición, estamos hablando de lo que se conoce como **cinemática inversa (*Inverse Kinematics*, IK)**. Es decir, dada una posición y orientación deseadas para el último eslabón de una cadena (efector), establecer las transformaciones requeridas para el resto de la cadena.

La IK ofrece mucha más interactividad y una flexibilidad añadida que permite animar un personaje de forma completa sin necesidad de utilizar animaciones pregrabadas.

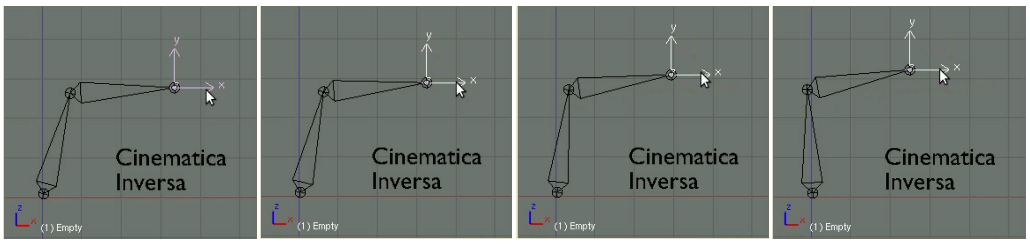


Figura 10: Cinemática Inversa

Repasando el caso de estudio, se plantea mover el brazo (o cualquier otra extremidad) con el ratón de forma arbitraria para alcanzar un punto determinado (y de no alcanzarse, aplicarse la elasticidad de los miembros); por ello se desprende que la parte interesante de estas técnicas para el caso de estudio es la cinemática inversa (se calculan las posiciones y orientaciones de

los huesos a partir del punto en el que hayamos pinchado con el ratón) y no la directa, que implicaría rotar con el ratón explícitamente y por separado cada uno de los huesos.

Grados de libertad (DOF)

Dada una figura articulada, cada una de las articulaciones conforma el número de grados de libertad para el siguiente objeto de la jerarquía. Por ejemplo, una suspensión de un coche tiene sólo un grado de libertad traslacional para el elemento final de la cadena (articulación prismática) [2][24].

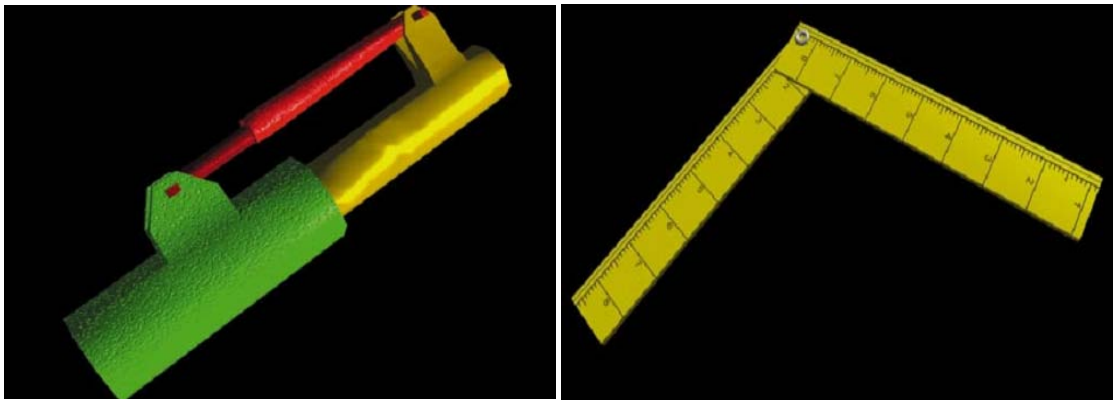


Figura 11: ejemplos de grados de libertad traslacional (izda.) y rotacional (dcha.)

Por otro lado, una articulación esférica (o enartrosis) tiene libertad en los ejes rotacionales x, y, z pero no puede desplazarse. La figura 11b representa un único grado de libertad rotacional.

Cada vez que agregamos un grado de libertad, los cálculos se vuelven mucho más complejos.

En realidad, la mayoría de articulaciones de un personaje tienen más de un solo grado de libertad. Por ejemplo, la muñeca suele poder rotarse hasta cierto punto en los ejes *x*, *y* y *z*. Esto supone tres grados de libertad sólo para este hueso.

Al calcular la cinemática inversa para un sistema, lo que se está haciendo es resolver un sistema de ecuaciones no lineales. Cada grado de libertad adicional aumenta la complejidad del problema, lo que significa que si es posible limitar el sistema de alguna forma se estarían simplificando enormemente los cálculos.

4.3. Curvas, superficies y volúmenes de Bézier

Curvas de Bézier

En el campo matemático del análisis numérico o geométrico, se conoce como curva de Bézier a un tipo de curva paramétrica de especial relevancia en el ámbito de los gráficos por computador. Las curvas de Bézier se hicieron populares en 1962 gracias al ingeniero francés Pierre Bézier, quien las utilizó en el diseño de componentes de automóviles. Fueron desarrolladas en 1959 por Paul de Casteljau usando el algoritmo que lleva su nombre, un método numéricamente estable para evaluar dichas curvas.

En el ámbito de los gráficos vectoriales, las curvas de Bézier suponen una importante herramienta empleada para modelar curvas suaves que se pueden escalar de forma indefinida. Su uso también está extendido para ciertos tipos de animación. A menudo se utilizan este tipo

de curvas para describir o controlar la velocidad en el tiempo de un objeto que se mueve de un punto a otro [40][41].

La definición paramétrica de la curva de Bézier es una función $Q(u)$, la cual está definida en términos del parámetro *u* donde *0 < u < 1*. Cuando *u* varía de 0 a 1 en el dominio de la función, para una curva cúbica el rango de la función encuentra una serie puntos intermedios entre P_0 y P_3, los cuales resultan de mezclas y escalas de los 4 puntos de control P_0, P_1, P_2 y P_3. Esto es, cada punto en la curva es determinado escalando cada punto de control por un polinomio cúbico conocido como base o función de mezcla.

La ecuación paramétrica $Q(u)$ tiene la forma de las ecuaciones 1 y 2.

$$Q(u) = \sum_{i=0}^{x} \boldsymbol{P_i} B_{i,x}(u), \qquad u \in [0,1] \tag{1}$$

$$B_{i,x}(u) = \binom{x}{i} e^i (1-u)^{x-1}, \qquad i = 0 \dots n \tag{2}$$

Si se fija el grado de la curva a 3, la ecuación 2 puede expandirse de tal modo que se obtienen las cuatro ecuaciones para la expresión 1, dependientes únicamente del parámetro *u*, como se puede ver en la ecuación 3.

$$\begin{gathered} B_{i,x}(u) = \binom{x}{i} e^i (1-u)^{x-1}, \qquad i = 0 \dots n \\ B_{i,3}(u), \qquad i = \{0,1,2,3\} \\ B_{0,3}(u) = (1-u)^3 \\ B_{1,3}(u) = 3u(1-u)^2 \\ B_{2,3}(u) = 3u^2(1-u) \\ B_{3,3}(u) = u^3 \end{gathered} \tag{3}$$

Las expresiones de la ecuación 3, funciones de tercer grado de Bézier, pueden utilizarse para expandir la ecuación 1. El resultado es una función única que se puede emplear para evaluar la posición en una curva de Bézier cúbica para cualquier valor del parámetro *u*:

$$\begin{aligned} Q(u) = \sum_{i=0}^{3} P_i B_{i,3}(u) = {} & P_0 B_{0,3}(u) + P_1 B_{1,3}(u) + P_2 B_{2,3}(u) + P_3 B_{3,3}(u) = P_0(1-u)^3 \\ & + P_1 3u(1-u)^2 + P_2 3u^2(1-u) + P_3 u^3 \end{aligned} \tag{4}$$

Se puede dibujar la curva resultante calculando el valor de la función en un determinado intervalo, y luego conectando dichos puntos con líneas rectas. La figura 12 representa una curva de Bézier de tercer grado, con los cuatro puntos de control P1, P2, P3 y P4.

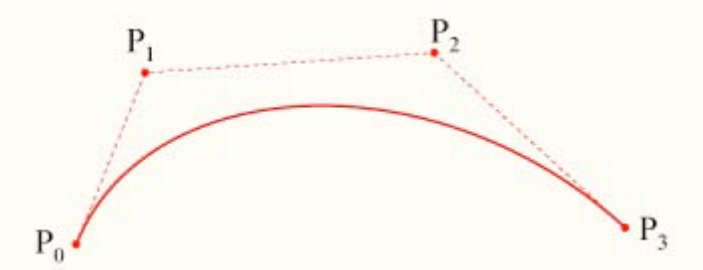


Figura 12: Curva de Bézier de grado 3

Superficies de Bézier

Dada la notación de una curva, es sencillo extenderla para representar una superficie. De la forma general de una curva de Bézier expuesta en la ecuación 1, una superficie paramétrica es una función de dos parámetros, denominados *u* y *v*, como se muestra en la ecuación 5:

$$Q(u,v) = \begin{bmatrix} X(u,v) \\ Y(u,v) \\ Z(u,v) \end{bmatrix}, \qquad u\epsilon[0,1], v\epsilon[0,1] \quad (5)$$

Una superficie de Bézier simple (de grado 3, o cúbica) se conoce a menudo como “parche”; la figura 13 muestra un parche de Bézier definido en dos dimensiones.

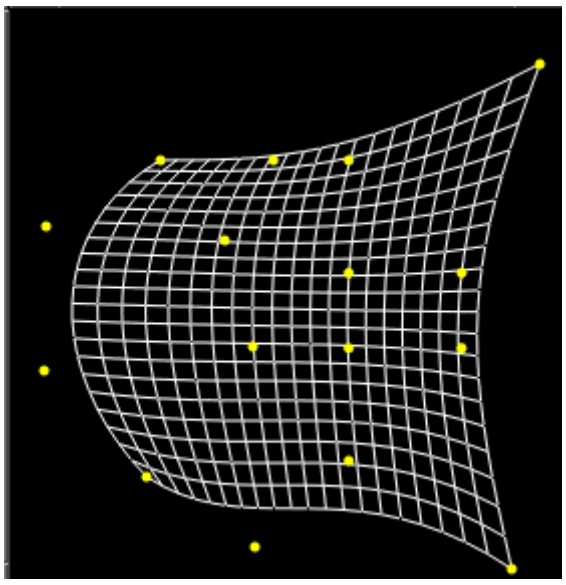


Figura 13: Parche Bézier

De nuevo, los puntos de control definen la forma del parche; éste viene definido por 16 puntos de control. La ecuación 6 representa un parche de Bézier:

$$Q(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} P_{i,j} B_{i,3}(u) B_{j,3}(v) \quad (6)$$

El conjunto de puntos de control *P* se extiende de 4 vectores para una curva a una matriz de 4x4 vectores para una superficie.

La forma más sencilla de imaginar una superficie paramétrica es pensar en utilizar las cuatro filas de la matriz para dibujar cuatro curvas horizontales, y las cuatro columnas para dibujar cuatro curvas verticales. Juntas, estas curvas producen una malla de cuadriláteros.

Volúmenes de Bézier

Una vez expuesto el concepto de superficie de Bézier, de nuevo extender esta teoría a volúmenes simplemente supone aumentar el número de puntos de control y añadir otro parámetro a la ecuación de Bézier. El conjunto de puntos de control *P* ahora contiene 64 vectores en una matriz 4x4x4, y la curva vendrá definida en términos de tres parámetros, como se muestra en la ecuación 7:

$$Q(u,v,w) = \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} P_{i,j,k} B_{i,3}(u) B_{j,3}(v) B_{k,3}(w) \qquad (7)$$

Una de las estructuras más básicas que podemos construir con un volumen de Bézier es una forma cúbica. Para ello disponemos los puntos de control para formar una cuadrícula de 4x4x4. En este caso los puntos de control, dibujados en un espacio 3D, tendrían el aspecto de la figura 14.

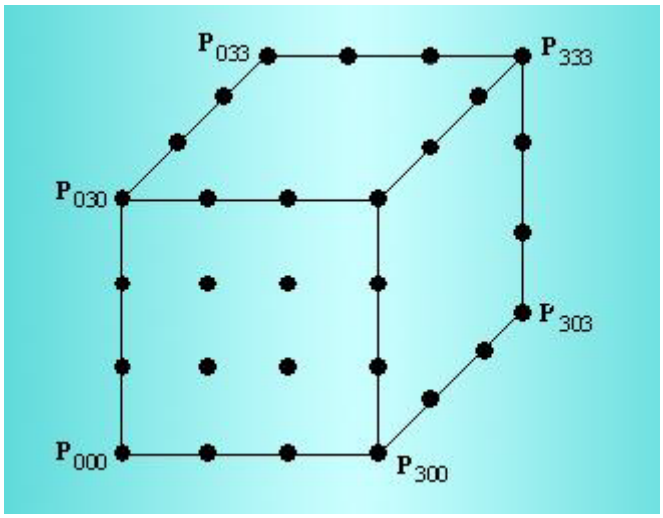


Figura 14: Volumen cúbico de Bézier

Pero un volumen de Bézier no define simplemente una superficie de seis caras, sino que se trata de un volumen real en el que los puntos de control pueden alterarse libremente para producir gran variedad de volúmenes. Al mover los puntos del volumen, los ejes inicialmente rectos se deformarán dando lugar a curvas de Bézier.

4.4. Osciladores Armónicos

Como base teórica para la aplicación de movimientos de tipo elástico, se ha empleado la teoría de osciladores armónicos [42].

Un oscilador armónico es un sistema que, al desplazarse desde su posición de equilibrio, experimenta una fuerza restablecedora (*F*) proporcional al desplazamiento (*x*) tal y como enuncia la Ley de Hooke (ec. 8).

$$F = -kx, \qquad k \geq 0 \qquad (8)$$

Si *F* es la única fuerza que actúa sobre el sistema, entonces éste se denomina oscilador armónico simple, y se rige por el movimiento armónico simple: oscilaciones sinusoidales alrededor del punto de equilibrio con una amplitud y frecuencia constantes.

Si en el sistema actúa también una fuerza friccional (o de amortiguación) proporcional a la velocidad, el oscilador se denomina entonces oscilador amortiguado.

Si por el contrario actúa sobre el sistema una fuerza externa dependiente del tiempo, el sistema se conoce como oscilador forzado.

Oscilador Armónico Simple

Un oscilador armónico simple es un oscilador que no es forzado ni amortiguado; su movimiento es periódico, repitiéndose de forma sinusoidal con una amplitud constante *A*. El movimiento armónico simple es un modelo matemático que puede emplearse para modelar gran variedad de movimientos, como el de un péndulo o un sistema de muelles y masas (figura 15).

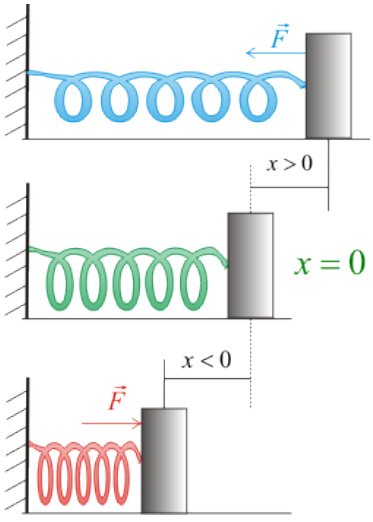


Figura 15: Oscilador Armónico Simple

Además de por su amplitud, el movimiento de un oscilador armónico simple se caracteriza por su período *T*, el intervalo temporal de una única oscilación, su frecuencia, *f* (recíproca del período $f = \frac{1}{T}$), y su fase, *φ*, la cual determina el punto de partida de la onda sinusoidal. Período y frecuencia son constantes determinadas por el conjunto del sistema, mientras que la amplitud y la fase se determinan por las condiciones iniciales (posición y velocidad) del sistema. La ecuación 9 describe el movimiento armónico simple.

$$x(t) = A * \sin(2\pi f t + \phi) \qquad (9)$$

La ecuación 10 representa la fórmula general para la fuerza diferencial de un objeto de masa *m* experimentando un movimiento armónico simple.

$$F = m\frac{d^2x}{dt^2} = -kx \qquad (10)$$

Donde *k* es la constante elástica, que relaciona el desplazamiento del objeto con la fuerza aplicada al mismo.

Oscilador armónico amortiguado

Nótese que la oscilación descrita en el apartado anterior se prolongaría indefinidamente en el tiempo (la sinusoide que describe la posición no converge a cero en ningún momento). En osciladores reales, existirían fuerzas externas que reducirían el movimiento del sistema.

Estas fuerzas pueden ser constantes, pero siempre con signo tal que se frene el movimiento. Es el caso de rozamientos secos: la fuerza no depende ni de la velocidad ni de la posición. Otra situación que se produce en la realidad es que la fuerza sea proporcional a la velocidad elevada a una potencia, entera o no. Así sucede cuando la fuerza que frena proviene de la viscosidad o de las pérdidas aerodinámicas. Se tratará únicamente el caso más simple, es decir, cuando la fuerza sea proporcional a la velocidad. En este caso la fuerza responderá a la ecuación 11.

$$F_f = -bv = -b\frac{dy}{dt} \qquad (11)$$

Donde *b* es un coeficiente que mide el amortiguamiento debido a la viscosidad. Si *b* es pequeño, el sistema está poco amortiguado. Nótese el signo negativo que indica, como antes, que si la velocidad es positiva, la fuerza tiene la dirección opuesta a la velocidad. Con este término complementario la ecuación diferencial del sistema es:

$$m\frac{d^2y}{dt^2} = -ky - b\frac{dy}{dt} \qquad (12)$$

Se trata de una ecuación diferencial ordinaria, lineal, de segundo orden (contiene derivadas segundas) y homogénea (no hay término independiente de y). Tiene tres tipos de soluciones según el valor de $b^2 - 4km$:

- Si $b^2 - 4km > 0$ el sistema está sobreamortiguado (amortiguamiento fuerte o supercrítico)
- Si $b^2 - 4km = 0$ el sistema tiene amortiguamiento crítico.
- Si $b^2 - 4km < 0$ el sistema oscila con amplitud decreciente (amortiguamiento débil o subcrítico)

5. Versión 1: Base del marco de trabajo para la representación de objetos 3D

5.1. Objetivos

En la primera iteración del modelo incremental se plantea el desarrollo de un marco de trabajo base para la representación de objetos 3D de forma jerárquica y completa.

OpenGL provee de una API a bajo nivel para la creación de modelos o mallas de polígonos a partir de primitivas (vértices, normales, texturas y coordenadas de textura, principalmente). La primera versión del programa proporcionará una interfaz que abstraiga de y trabaje sobre esa API para simplificar la carga y dibujado de objetos cargándolos a partir de un formato de modelado ampliamente extendido, así como los elementos básicos que rodean a estos objetos: iluminación, materiales, texturas, etc.

Los objetos se organizarán de forma jerárquica a modo de nodos, de tal manera que trasladar un objeto supondrá trasladar a todos los elementos que caigan bajo su posición en la jerarquía. Se permitirá asimismo la gestión de la ventana del programa y una interfaz de manejo del mismo.

5.2. Análisis

5.2.1. Requisitos Funcionales

Código requisito	Nombre requisito	Descripción de requisito
RV01F01	Representación y dibujado de objetos	Se debe poder representar, especificando la información de sus vértices, polígonos o las primitivas pertinentes, objetos 3D visualizadles en el espacio de OpenGL.
RV01F02	Carga de objetos generados de forma externa	La representación de objetos debe incluir la posibilidad de cargar la información de vértices, polígonos y texturas a partir de un formato exportable.
RV01F03	Texturizado	Deben poder cargarse imágenes en formatos extendidos (.jpg, .png) y a partir de unas coordenadas de textura explícitas, aplicar el coloreado de los píxeles en un polígono según dicho texturizado.
RV01F04	Transformación de objetos	Se debe proporcionar una interfaz para la transformación básica de los objetos cargados en la escena; a saber: traslación, rotación y escalado, a través de las primitivas que para tal efecto define OpenGL.

RV01F05	Gestión de ventanas y escenas	Se proporcionará una interfaz para la gestión de distintas escenas y una única ventana para su visualización en el operativo.

Tabla 5: Requisitos funcionales de la versión 1

5.2.2. Requisitos no funcionales

Código requisito	Nombre requisito	Descripción de requisito
RV01NF01	Jerarquía de objetos	Los objetos cargados en la escena 3D a mostrar en pantalla deben poder estructurarse de forma jerárquica, de tal modo que transformar (rotar, escalar, desplazar) un objeto suponga hacerlo con los objetos inferiores en la jerarquía.
RV01NF02	Flexibilidad de objetos	Podrán cargarse en la escena objetos simples, personajes con malla ósea, con deformación elástica, etc. Cualquier tipo de objeto definido con posterioridad debe adaptarse al formato de la escena (interfaz). Se abre el desarrollo a la permisión de objetos construidos en base a otras primitivas que el uso simple de vértices y polígonos (ej. B-Splines).
RV01NF03	Eficiencia	La gestión y representación de objetos deberá ser lo más eficiente y sencilla posible en aras de liberar al procesador de carga de trabajo para dedicarle a posteriores tareas de deformación.
RV01NF04	API Sencilla	La API de trabajo para el programador (la interfaz de gestión del “motor”) debe ser sencilla y suponer realmente una abstracción y no una simple réplica de la ofrecida por OpenGL.

Tabla 6: Requisitos no funcionales de la versión 1

Para el primer incremento del desarrollo del software a la hora de realizar el análisis muchas de las posibilidades para afrontar el desarrollo vienen de algún modo impuestas por la propia definición de alcance del proyecto.

Ocurre así, por ejemplo, con el uso de OpenGL como API (en detrimento de otras como la muy extendida DirectX), el cual viene indicado como base en la descripción del proyecto, de tal modo que de algún modo se impone esta opción. Ello no necesariamente implica que la decisión se haya tomado a la ligera; el uso de OpenGL, como se ha visto, se ha escogido frente

a su principal competidor por causas de relevancia como son la experiencia del desarrollador con el mismo y la abundante documentación existente al respecto; motivada principalmente porque la gran mayoría de artículos existentes del sector se basan en este estándar para el desarrollo de sus paradigmas.

Por otro lado, OpenGL es un estándar abierto, una filosofía que casa con el paradigma de desarrollo de una aplicación orientada a la formación del estudiante o desarrollador, pudiendo pues utilizarse el código para su publicación en la red u otros medios en aras de motivar el aprendizaje de otras personas en el futuro.

De cualquier modo, su condición de estándar abierto no debe provocar su desconfianza en cuanto a su potencia; hoy por hoy OpenGL ofrece un abanico de posibilidades gráficas completo y a la par con su homólogo DirectX, de modo que no existen motivos para condicionar el rechazo hacia el mismo y la elección de uno u otro paradigma vendrá indicada en cada caso más por preferencias o razones de carácter pragmático (soporte de uno u otro para determinada plataforma, por ejemplo) que por las capacidades de uno u otro.

5.2.3. El formato OBJ

En cuanto al formato de los objetos 3D se ha escogido el Wavefront OBJ frente a otros como 3DS, MAX, Maya, etc. por la sencillez de su propuesta al tratarse de objetos en texto plano con una exposición bastante explícita de la información, así como unas normas de sintaxis claras, flexibles y fácilmente acatables. Asimismo aun tratándose de un formato creado por una empresa comercial ha terminado por convertirse en un estándar abierto y soportado por prácticamente todas las aplicaciones de desarrollo 3D existentes en el mercado.

El formato .obj de Wavefront [8] se creó inicialmente para la suite de animación Advanced Visualizer, pero actualmente ha sido adoptado por toda la comunidad 3D, por lo que se suele considerar un estándar universalmente aceptado.

OBJ representa únicamente la geometría 3D, esto es, la posición de cada vértice, la posición UV de cada vértice de textura de coordenadas, las normales y las caras que constituyen cada polígono, definidas como una lista de vértices y vértices de textura, dejando la definición de materiales y los mapas de textura para otros recursos externos.

Los ficheros OBJ son de texto plano, y su información se describe de la siguiente forma:

`# texto`
: La línea representa un comentario.

`v` *`float float float`*
: Posición de un vértice único en el espacio. Al primer vértice de la lista se le asigna el índice 1, y los vértices subsiguientes se numeran secuencialmente.

`vn` *`float float float`*
: Representa una normal. Al igual que los vértices, la primera en la lista tiene índice 1 y las demás se numeran secuencialmente.

`vt` *`float float`*
: Coordenada de textura UV. Se numeran de igual forma que las normales o vértices.

`f int int int`
`f int/int int/int int/int`
`f int/int/int int/int/int int/int/int`

Representa una cara poligonal. Los números son índices dentro de los arrays de vértices, coordenadas de texturas y normales, respectivamente. Se puede omitir alguno de los números, por ejemplo, las coordenadas de textura, si estas no están presentes

`mtllib [archivo .mtl]`

Indica una referencia a un archivo en formato MTL que contendrá los materiales asociados a la geometría descrita.

`o [nombre objeto]`

Indica un objeto (o sub-objeto) dentro de la geometría descrita.

`g [nombre grupo]`

Indica un grupo de polígonos dentro de la geometría descrita.

`usemtl [nombre material]`

Indica el material a utilizar en la subsiguiente geometría.

5.2.4. El formato MTL

En concordancia con la elección del formato OBJ para la geometría, se ha escogido el formato MTL para los materiales asignados a cada una de las partes de la misma, tales como respuesta a la iluminación, color o texturas. Este tipo de archivo es referenciable desde un fichero OBJ y contiene a su vez referencias a las imágenes de las texturas.

`newmtl [nombre material]`
Indica la definición de un nuevo material

`Ka float float float`
Define la componente de color ambiente del material en RGB.

`Kd float float float`
Define la componente de color difusa del material en RGB

`Ks float float float`
Define la componente de color especular del material en RGB.

`d alpha`
Define la transparencia del material (0-1).

`Ns float`
Define el brillo del material.

`illum int`
Define el modelo de iluminación empleado por el material (1 – sin brillos especulares, 2- con brillos especulares).

`map_Kd [archivo_textura.ext]`
Referencia un archivo que contiene un mapa de texturas.

5.2.5. Formato de imágenes

Para la carga de imágenes de texturas y bump-maps, dado que el análisis 2D de estos objetos no entra dentro de los objetivos de estudio, se ha utilizado una librería externa llamada "*32-bit BGRA WIN32 device independent bitmap (DIB) class*", la cual define una clase *Bitmap* que encapsula todos los métodos y atributos necesarios para la carga de imágenes en formatos BMP, GIF, JPEG, PNG, TIFF y TGA a través del subcomponente GDI+ de Windows.

La clase permite, una vez cargada la imagen desde el archivo externo, realizar múltiples operaciones para el acceso y manipulación de los datos de píxeles, como clonar, pintar sobre la imagen y guardarla, modificar su tamaño, rotarla o invertirla vertical y horizontalmente, y por supuesto acceder directamente a la matriz de píxeles descrita por la imagen.

5.3. Diseño

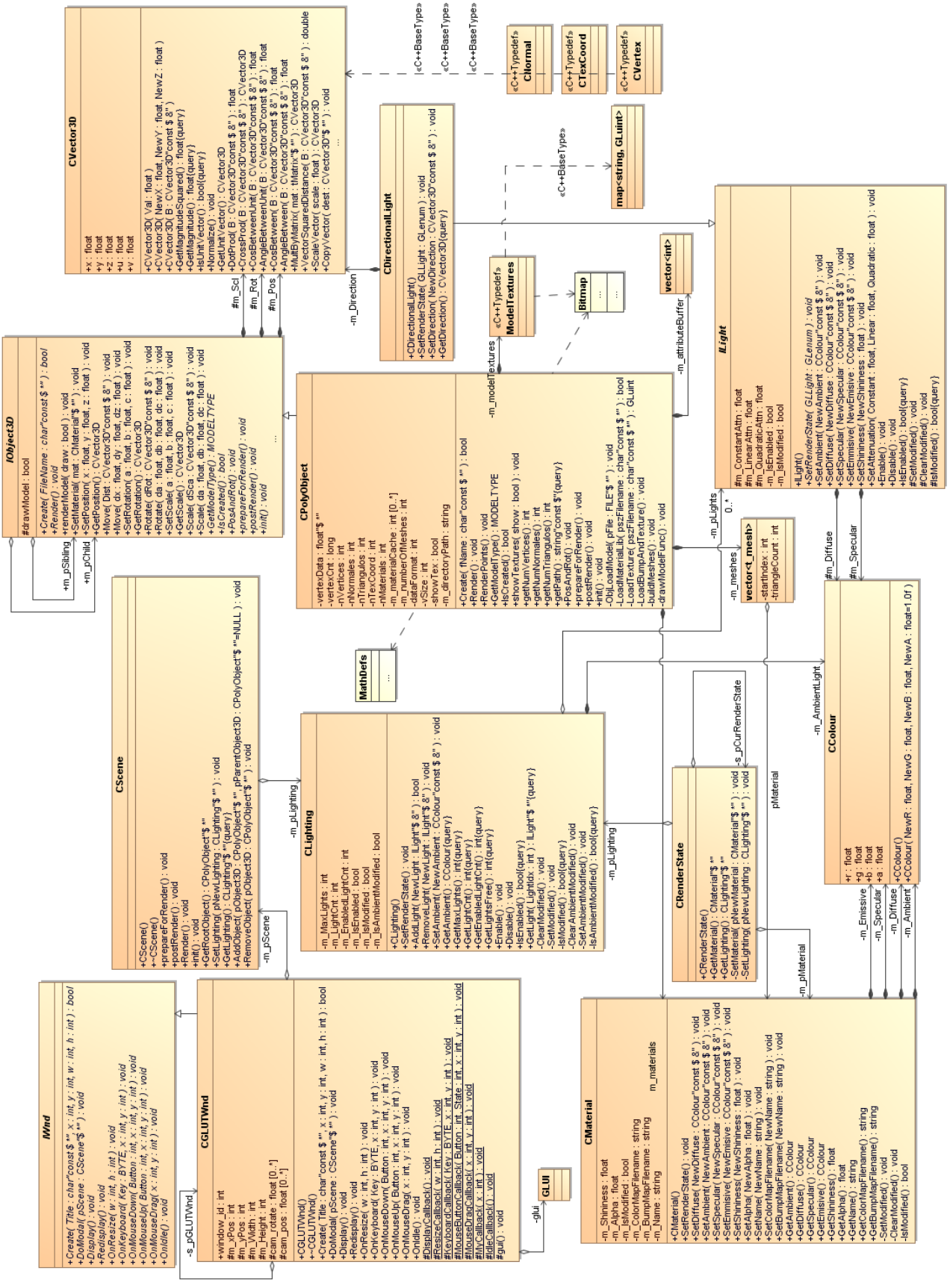


Figura 16: Diagrama de objetos para la versión 1

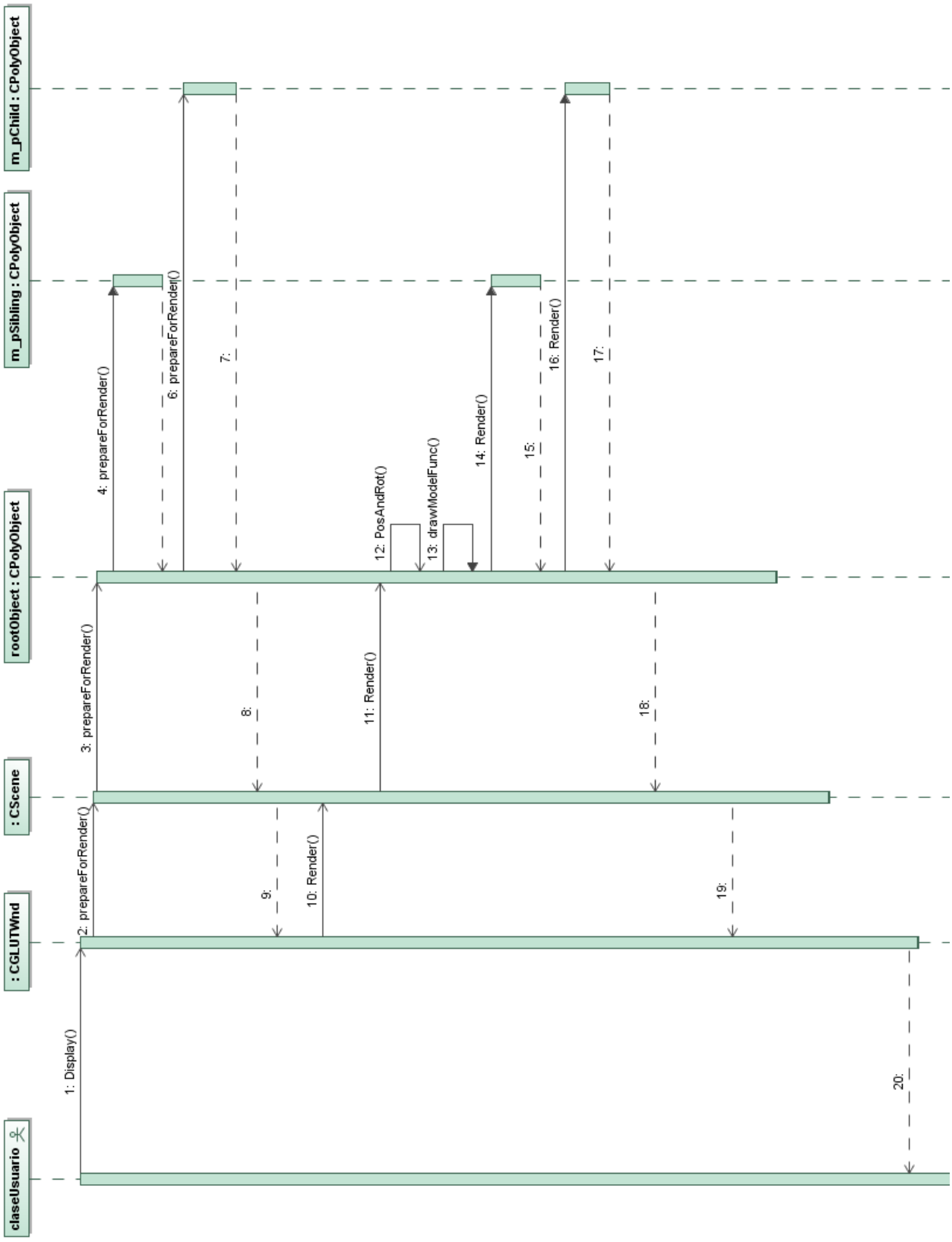


Figura 17: Diagrama de secuencia para el caso de uso renderizar escena, versión 1

5.4. Implementación

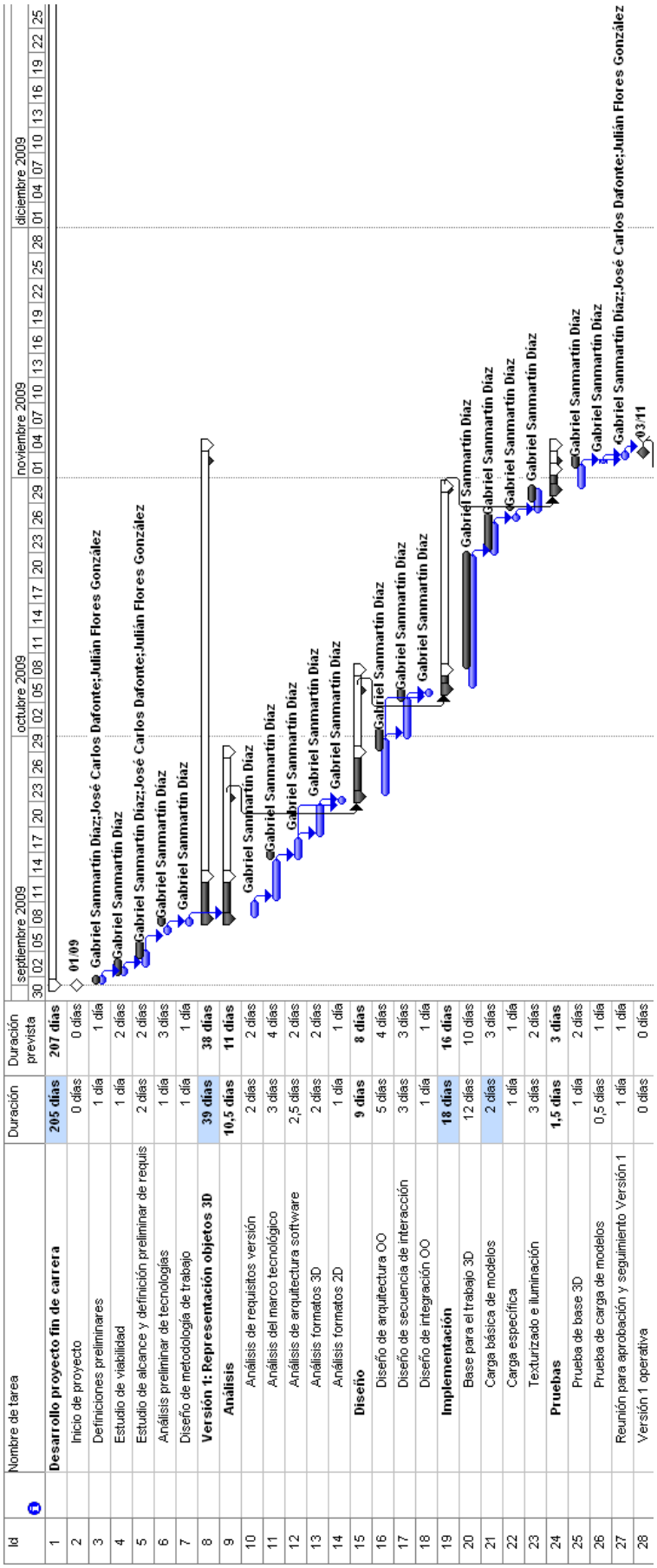
No existen detalles relevantes para la implementación de esta iteración.

5.5. Seguimiento del proyecto

Tras el desarrollo de este incremento el cómputo global es de un ahorro de 2 días respecto a lo planificado. Este adelanto viene motivado principalmente por la experiencia del programador con el ámbito de desarrollo (entornos y escenas 3D), lo que permitió reducir considerablemente las fases relativas al análisis de tecnologías existentes y de las técnicas de representación.

El ahorro en tiempos permitió tener una holgura temporal suficiente para compensar el retraso ocasionado en las fases de diseño (por la complejidad del diagrama desarrollado) e implementación de la base, debido a la poca práctica del programador con el lenguaje empleado, C++, para el que hubo de dedicarse más tiempo del previsto.

La entrega de la versión inicial operativa, por tanto, se realizó el día 3 de noviembre de 2009, dos días antes de la fecha de entrega prevista.



Id	Nombre de tarea	Duración	Duración prevista
1	**Desarrollo proyecto fin de carrera**	**205 días**	**207 días**
2	Inicio de proyecto	0 días	0 días
3	Definiciones preliminares	1 día	1 día
4	Estudio de viabilidad	1 día	2 días
5	Estudio de alcance y definición preliminar de requis	2 días	2 días
6	Análisis preliminar de tecnologías	1 día	3 días
7	Diseño de metodología de trabajo	1 día	1 día
8	**Versión 1: Representación objetos 3D**	**39 días**	**38 días**
9	**Análisis**	**10,5 días**	**11 días**
10	Análisis de requisitos versión	2 días	2 días
11	Análisis del marco tecnológico	3 días	4 días
12	Análisis de arquitectura software	2,5 días	2 días
13	Análisis formatos 3D	2 días	2 días
14	Análisis formatos 2D	1 día	1 día
15	**Diseño**	**9 días**	**8 días**
16	Diseño de arquitectura OO	5 días	4 días
17	Diseño de secuencia de interacción	3 días	3 días
18	Diseño de integración OO	1 día	1 día
19	**Implementación**	**18 días**	**16 días**
20	Base para el trabajo 3D	12 días	10 días
21	Carga básica de modelos	2 días	3 días
22	Carga específica	1 día	1 día
23	Texturizado e iluminación	3 días	2 días
24	**Pruebas**	**1,5 días**	**3 días**
25	Prueba de base 3D	1 día	2 días
26	Prueba de carga de modelos	0,5 días	1 día
27	Reunión para aprobación y seguimiento Versión 1	1 día	1 día
28	Versión 1 operativa	0 días	0 días

Figura 18: Seguimiento del proyecto a la entrega de la versión 1

6. Versión 2: Deformación Ósea

6.1. Objetivos

Traslación, Rotación o Escalado son las tres operaciones básicas de transformación de objetos en el espacio 3D, y son accesibles en OpenGL mediante primitivas (glTranslate, glRotate, glScale) que trabajan directamente sobre las matrices de transformación operables por las tarjetas gráficas 3D. En la base de desarrollo diseñada para la versión anterior estas operaciones se simplifican y encapsulan para cada objeto y su jerarquía inferior mediante las operaciones *rotate()*, *scale()* o *translate()* de la interfaz *IObject3D*.

Estas tres transformaciones básicas se realizan de forma global sobre los objetos afectados por las matrices de transformación. Sin embargo, para el caso de un personaje 3D, es casi siempre necesario que éste mueva únicamente partes localizadas de su cuerpo (ahí reside la base de la animación). Utilizar matrices específicas para vértices o grupos de vértices resulta claramente inviable, de tal modo que gestionar estas transformaciones locales requiere deformaciones específicas a nivel de vértice, que hacen que la malla resultante sea diferente que la utilizada originalmente.

A la hora de trabajar con personajes (u otros objetos) 3D en tiempo real, por lo tanto, mostrar animaciones con los mismos comporta representar en cada fotograma las nuevas posiciones de cada uno de los píxeles que conforman dicho objeto, con los correspondientes cálculos geométricos para hallar el color de cada uno de estos puntos, comprobando en cada momento qué objetos o vértices dentro de los mismos se hallarán visibles respecto a la cámara definida en un instante determinado.

Lo habitual en objetos poligonales es trabajar sobre los vértices que definen el volumen del modelo 3D, calcular sus nuevas posiciones y hallar entonces los píxeles del rellenado del polígono que serán representados en su rasterización 2D hacia el buffer de representación (framebuffer).

Abordar este trabajo vértice a vértice para cada fotograma sin una base que lo simplifique resulta inviable (modelos de decenas de miles de vértices son todavía de bajo detalle), por tanto esta segunda iteración deberá intentar definir un sistema para la gestión simplificada de animaciones y el movimiento de objetos “vivos” y orgánicos: un sistema óseo de deformación.

6.2. Análisis

6.2.1. Requisitos Funcionales

Código requisito	Nombre requisito	Descripción de requisito
RV02F01	Sistema de representación de personajes articulados	Se proporcionará un sistema para la representación de un modelo 3D bajo el cual se construye una jerarquía reflejando las articulaciones del personaje y sus elementos móviles.
RV02F02	Carga de objetos a partir de archivo	Se creará un formato para la representación de estos

esqueletos.

Tabla 7: Requisitos funcionales de la versión 2

6.2.2. Requisitos no funcionales

Código requisito	Nombre requisito	Descripción de requisito
RV02NF01	Verificación de Restricciones de grados de libertad	Un esqueleto real posee ciertas limitaciones de grados de libertad. El sistema implementado deberá, de forma transparente, prevenir la violación de dichas restricciones de forma autóctona.

Tabla 8: Requisitos no funcionales de la versión 2

6.2.3. "Skeletal Deformation" (Deformación ósea)

Técnicas para la construcción de mallas

Para el caso concreto de personajes 3D en tiempo real, existen diversas formas de diseñar algoritmos para efectuar los cálculos de las posiciones de los vértices y hacerlos más intuitivos al programador, pero el trabajo sobre la malla 3D de los modelos suele basarse siempre en dos tipos básicos [18][2]:

- Una jerarquía de varios objetos
- Personajes con una única malla

Jerarquías de varios objetos

Los personajes de este tipo tienen un objeto individual por cada parte de su cuerpo, que se conectan entre sí mediante algún tipo de jerarquía. Aplicando las rotaciones y traslaciones a cada una de las diferentes partes, se puede crear una sensación fluida de movimiento. Podemos encontrar este tipo de personajes especialmente en los primeros juegos 3D tales como *Tomb Raider* o *Jedi Knight* (fig. 19).

Figura 19: Personajes formados por varios objetos

Ventajas de este método:

- Los personajes creados como una jerarquía de objetos individuales son más maleables: mover una extremidad supone rotar un objeto independiente del resto.
- Sólo es necesario almacenar la transformación por cada objeto y la información de los vértices una vez, por lo que los archivos de animación son más pequeños. Los requerimientos en memoria también son menores, pues basta con almacenar la orientación de las diferentes articulaciones en los huesos.
- Al organizar el personaje en una jerarquía, es relativamente sencillo crear animaciones nuevas. Si disponemos de capacidades de Cinemática Inversa o sistemas dinámicos, podemos generar animaciones nuevas "*on-the-fly*", simplificándose las colisiones y otras reacciones al entorno ("*Blended animation*"). Ésta es la principal ventaja de este método.

Desventajas:

- Dado que la posición real de cada vértice en el modelo viene determinada por la jerarquía, es necesario realizar cierto trabajo extra a la hora de representar el modelo en la pantalla. Cada objeto de la malla debe atravesar las transformaciones de la totalidad de la jerarquía superior a él. Esto comporta algo más de trabajo matemático por cada vértice. Por ejemplo, la malla de la mano izquierda debe recibir las transformaciones de caderas, pecho, antebrazo izquierdo, brazo izquierdo y finalmente la mano izquierda.
- En los puntos en que los objetos de la jerarquía se conectan entre sí aparecen huecos (fig. 20). Dado que dichos objetos son individuales, hay que considerar la forma en que se unen entre sí. Se pueden ocultar en cierta manera estas "uniones" modelando el objeto de forma específica, aunque no siempre es posible.

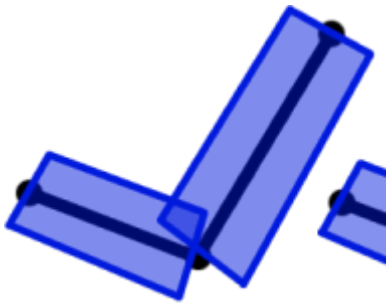


Figura 20: Unión de varios objetos al flexionar una articulación

Personajes de malla única

Como su nombre indica, son personajes creados con una única malla en lugar de objetos individuales. El personaje se anima cambiando directamente la posición de todos y cada uno de los vértices del modelo. Esto puede hacerse almacenando una tabla de posiciones u *offsets* para cada frame o simplemente alternando entre mallas.

Ventajas:

- Este tipo de personajes son visualmente más realistas que las jerarquías de múltiples objetos, en especial cuando se trabaja con un número reducido de polígonos. No existen junturas visibles y por tanto la malla es visualmente más compacta.
- Es más rápido. Dado que no es necesario transformar cada vértice a través de una jerarquía hay menos operaciones matemáticas necesarias en el proceso.

Desventajas:

- Los personajes son menos maleables; puesto que deben ser animados por un sistema externo, no es tan sencillo obtener animaciones únicas o específicas.
- Los archivos de animaciones son mayores.

Malla deformable única

La pregunta que surge ahora es si es posible combinar la flexibilidad de la jerarquía de objetos con los buenos resultados visuales de una malla única. La respuesta está en lo que se conoce como **malla deformable única** [23][26].

Consiste en definir un sistema de huesos "por debajo" de la malla única del personaje, de forma que el movimiento (rotación) de cada hueso tenga influencia sobre cada vértice en mayor o menor medida (peso). De esta manera, el esqueleto será una jerarquía de objetos separados (huesos), situado bajo una única malla (la que veremos finalmente) que se despliega sobre ellos.

Esta técnica se usa hoy en día en la práctica totalidad de juegos, y permite la flexibilidad y maleabilidad de los sistemas jerárquicos (pudiendo de nuevo gestionarse de manera relativamente sencilla la cinemática inversa y otros sistemas de animación "blended"), con los buenos resultados visuales de la malla única.

Como punto de partida, considérese la posibilidad de asociar cada vértice dentro de la malla con un único hueso. La influencia de dicho hueso tiene un efecto directo con el vértice asociado. Por tanto, si rotamos el hueso, los vértices que tenga asociados también rotarán, de la misma forma, sobre la posición base de dicho hueso.

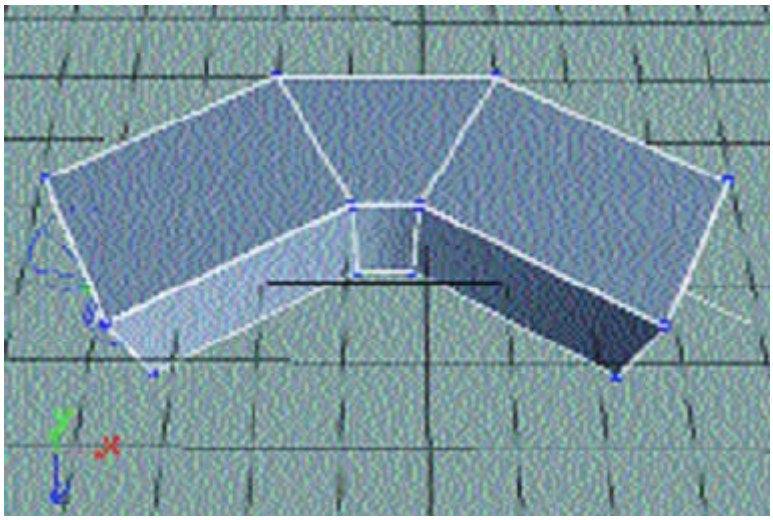


Figura 21: Unión de malla única sin ponderación de pesos

Puede verse el resultado de este proceso en la figura 21. En esta imagen, la jerarquía se compone de dos huesos, cada uno de los cuales tiene ocho vértices asociados. Aunque esta técnica crea una malla deformable sin junturas visibles y hace muy poco uso de procesador, tiene una desventaja. En el punto en que se conectan los dos huesos, la malla se estira visiblemente. Puede considerarse a este efecto como un mal menor para ciertas aplicaciones, pero lo cierto es que resulta poco realista.

Para resolver este problema basta añadir unos cuantos vértices al modelo y asignarles un "peso" individualmente. Es decir, que por cada vértice del modelo, se le asigna cierto porcentaje de influencia para cada hueso. Muchos vértices recibirán una influencia del 100% de un único hueso, mientras que otros estarán 50/50 bajo la influencia de dos huesos. Ajustando esta influencia, podemos suavizar el efecto mencionado. En la figura 22 se puede ver el efecto.

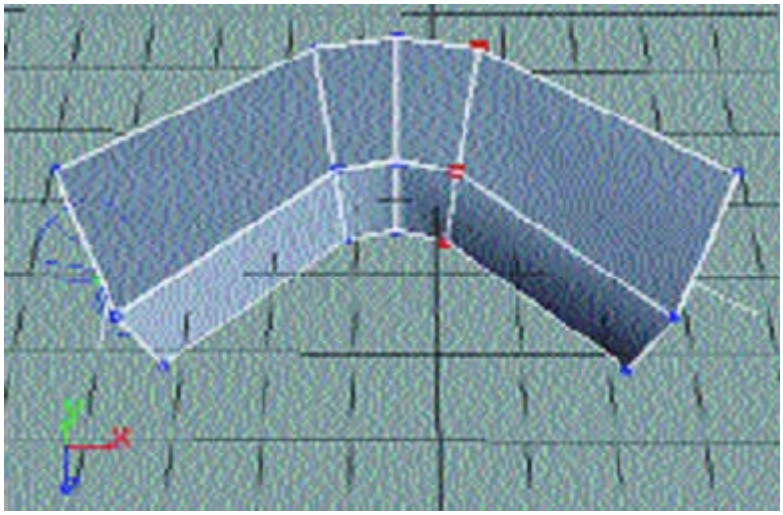


Figura 22: Unión de malla única con pesos ponderados

Debido a los cálculos necesarios para llevar a cabo la asignación de pesos, este sistema requiere algo más ciclos de procesador que su versión más básica. Sin embargo, la mejora de los resultados merece la pena frente a la pérdida de ciclos de reloj, despreciable además con los sistemas actuales.

6.3. Diseño

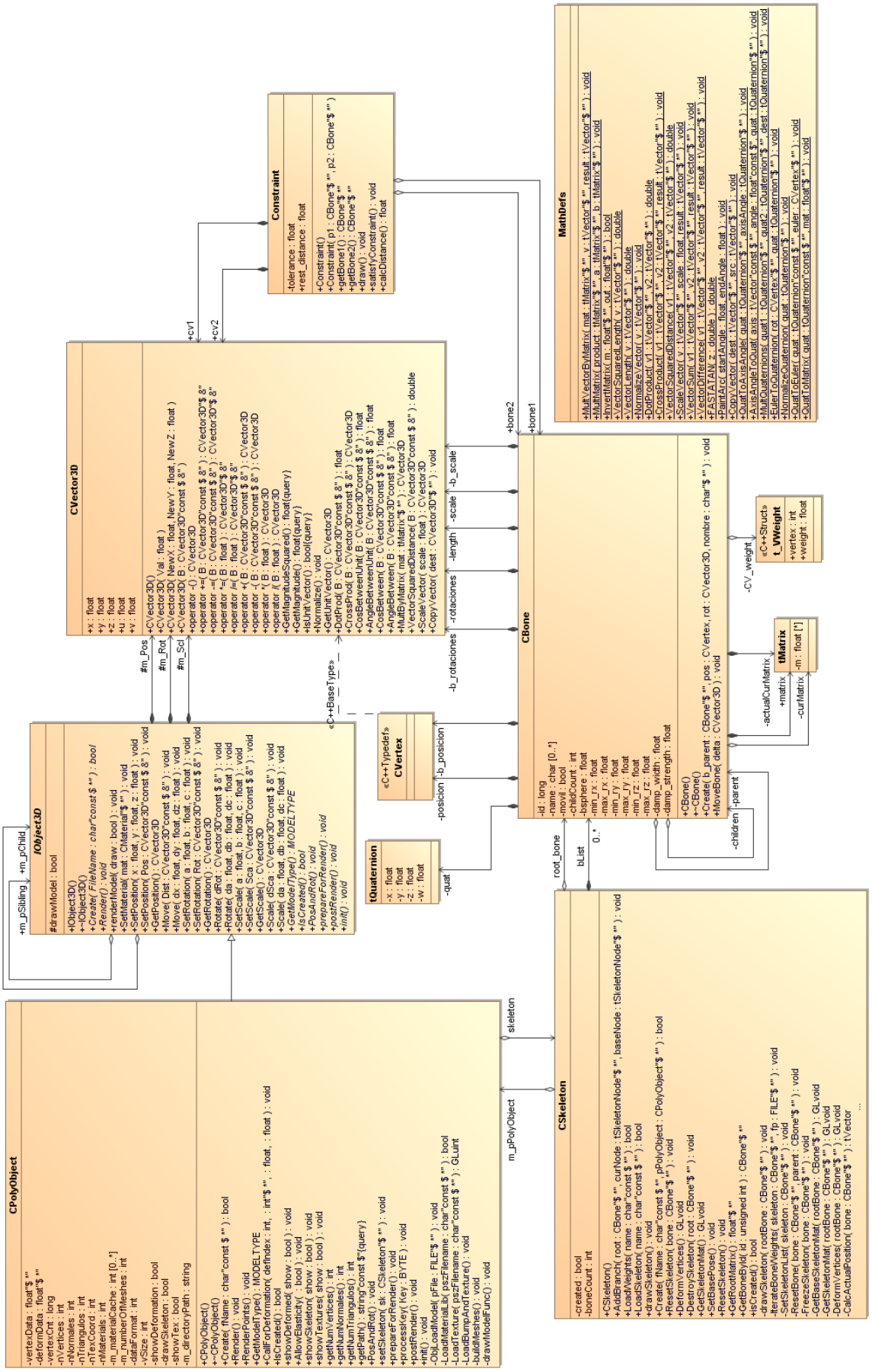


Figura 23: Diagrama de objetos para la versión 2

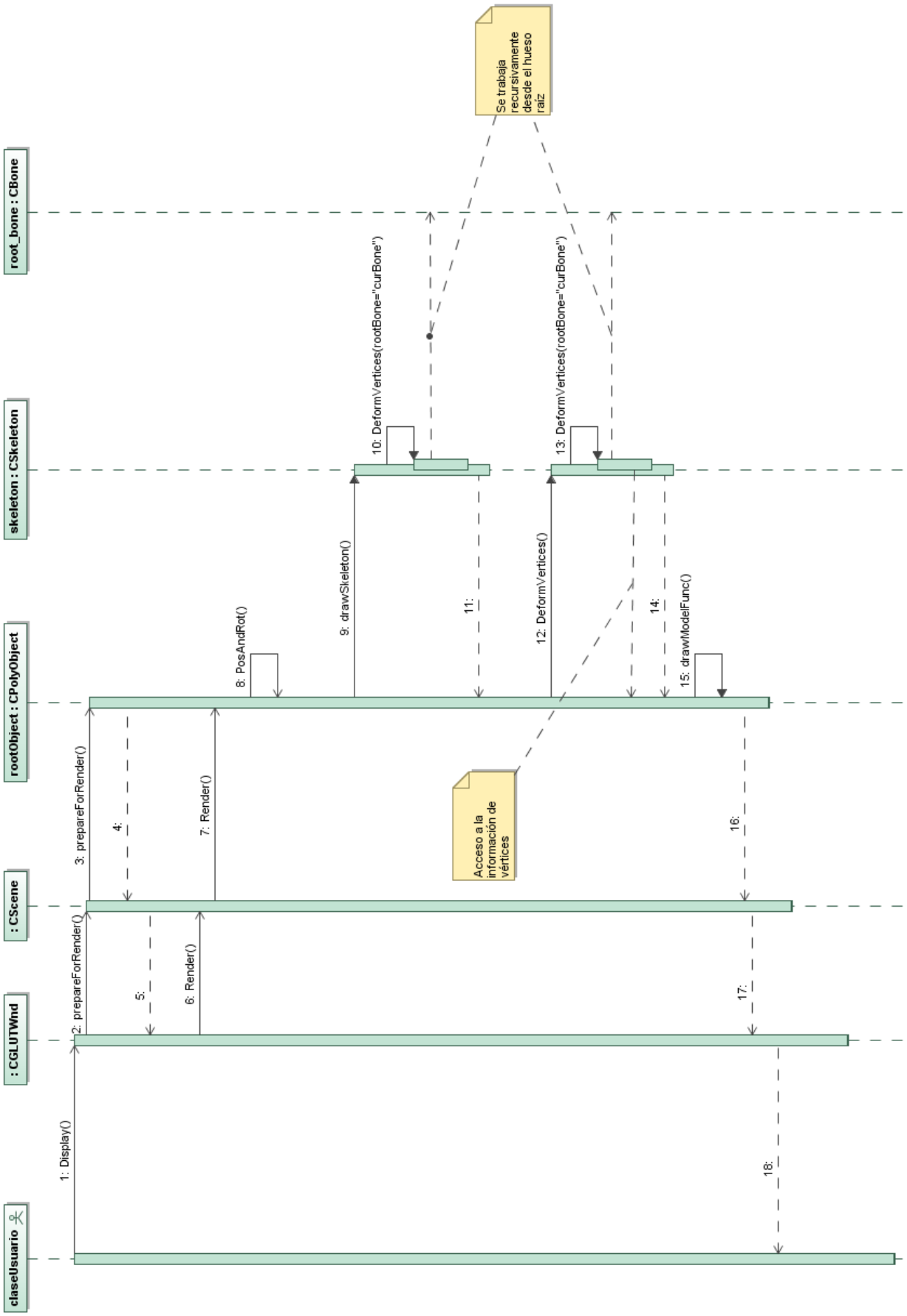


Figura 24: Diagrama de secuencia para el caso de uso "renderizar escena", versión 2

6.4. Implementación

6.4.1. Sistema óseo

Como se ha visto, para la representación de objetos rígidos efectuar transformaciones pasa por configurar la matriz de transformaciones y a continuación aplicarla sobre el objeto que se desea dibujar (se multiplica por su estado actual). Sin embargo, un personaje orgánico no ofrece resultados tan buenos cuando se compone de objetos rígidos; resulta mucho más realista visualmente crear una malla única y deformarla a través de un sistema óseo.

La desventaja es que este sistema requiere la manipulación directa de las coordenadas de los vértices.

En la figura 25 se puede ver un personaje en la llamada "bind pose" (pose inicial, [33]) y el esqueleto asociado al mismo. Para poder deformar la malla con dicho esqueleto, como se ha explicado, es necesario asignar cada vértice a un hueso o conjunto de huesos. Por ejemplo, todos los vértices de la cabeza deberían asignarse al hueso de la cabeza (supóngase que se asocian al 100% al hueso de la cabeza única y exclusivamente).

Puede determinar la posición y orientación del hueso de la cabeza en la pose inicial creando una matriz que represente la transformación necesaria para mover dicho hueso desde el origen de coordenadas a su posición dentro de la jerarquía. Pero la matriz de cada hueso depende de las matrices de los huesos que queden sobre ella en la jerarquía, de modo que es necesario atravesar ésta para obtener esta matriz **coordenadas_mundo->pose_inicial**.

Cada hueso *i* tendrá, por tanto, un sistema de coordenadas (matriz de transformaciones) asociado, T_i^B, cuyo propósito es trasladar un vértice, a partir del origen de coordenadas, a su posición en la pose inicial del personaje o "*bind pose*".

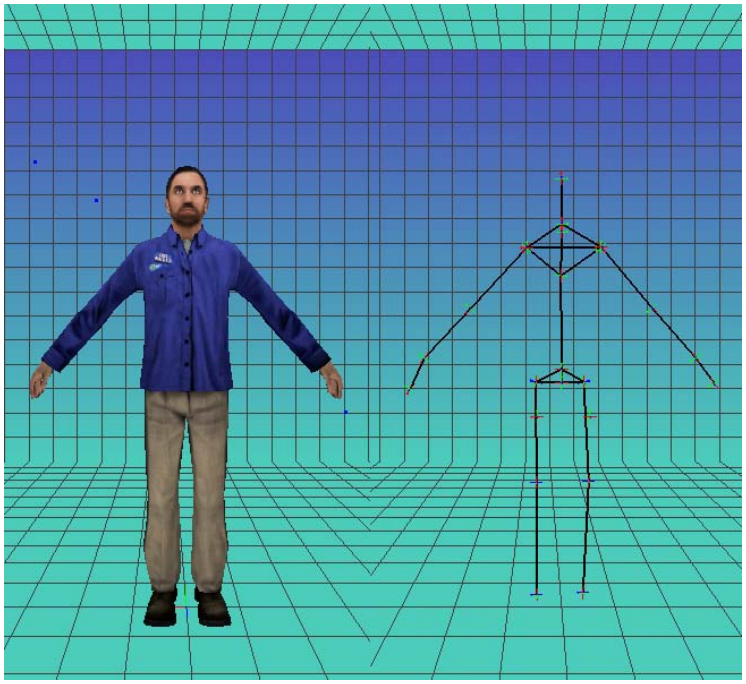


Figura 25: Personaje modelo (izda.) y su esqueleto (dcha.)

La matriz T_i^B puede aplicarse pues sobre un vértice para transformarlo hasta la posición del hueso *i* en su pose inicial. Sin embargo, al tomar la posición inicial de un personaje, es necesario saber cómo, dado un vértice de la malla, transformarlo de nuevo al origen de coordenadas. Afortunadamente, dado que seguimos trabajando con matrices, basta con invertir la matriz **coordenadas_mundo->pose_inicial** (T_i^B), para obtener **pose_inicial -> coordenadas_mundo** $(T_i^B)^{-1}$.

Esta será la matriz necesaria para mover cualquier vértice desde la posición de reposo al origen. Si ahora se desea mover dicho vértice (y todos los demás) a su posición final como la que se muestra en la figura 26, puede hacerse de la misma manera. Se atraviesa la jerarquía completa y se crea para cada hueso una matriz que tomará un vértice desde el origen y lo moverá a la posición del hueso asociado en ese frame de la animación. Es la matriz **coordenadas_mundo->pose_actual** (T_i^W).

Figura 26: Pose resultado de animar algunos huesos

Por tanto, para tomar un vértice en su posición de reposo y moverlo a la posición final, se seguirá el siguiente procedimiento:

$$\texttt{verticeMundo} = \texttt{vérticeModeloBase} * \left(T_i^B\right)^{-1}$$
$$\texttt{verticeDeformado} = \texttt{vérticeMundo} * T_i^W$$

Hay una optimización sencilla a todo esto; dado que dos matrices pueden multiplicarse para crear una matriz que suponga la suma de las transformaciones de ambas, es posible crear una matriz combinada (T_i^C) que haga todo lo necesario, como se muestra en la ecuación 13.

$$T_i^C = (T_i^B)^{-1} * T_i^W \qquad (13)$$

En conclusión, por cada vértice, simplemente debe multiplicarse éste por la matriz combinada T_i^C, asociada a su hueso. Esta es la clave para la deformación ósea. Si se desea que varios huesos tengan influencia sobre un vértice, esta fórmula simplemente debe escalarse por la cantidad de influencia (o peso) de cada hueso. Las matrices asociadas a cada hueso se actualizan, para cada fotograma, en la función *getSkeletonMat()* de la clase *Cskeleton.*

Todo este proceso se ilustra en la sucesión de imágenes 27 con el hueso de la mano del personaje utilizado. La figura 27a presenta una representación del esqueleto, en la cual el hueso de la mano se halla destacado con un círculo rojo.

La figura 27b representa el área del modelo 3D asociada al hueso de la mano, tras haberse multiplicado sus vértices por la matriz T_i^B. La siguiente figura (27c), "devuelve" dichos vértices al origen de coordenadas, multiplicándolos por $(T_i^B)^{-1}$.

A continuación, el esqueleto se mueve (a consecuencia de una animación), de tal forma que las posiciones resultantes de los huesos del brazo son las representadas por la figura 27d.

Para poder ubicar de nuevo la mano en la posición actual del hueso, basta multiplicar sus vértices por la nueva matriz T_i^W, tal y como se muestra en la figura 27e.

Este proceso se realiza una vez por cada frame.

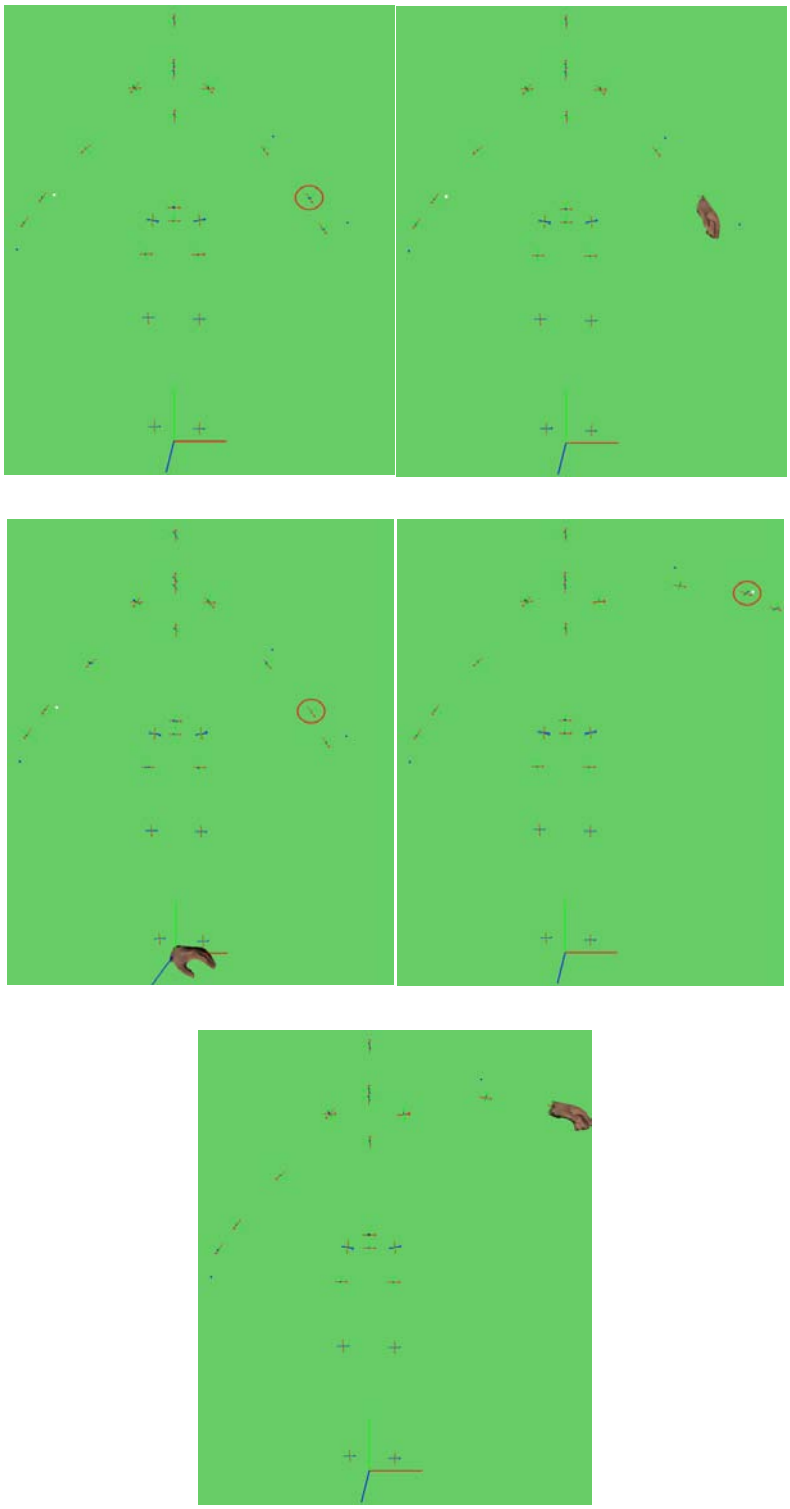


Figura 27: representación del esqueleto (a), vértices del modelo asociados al hueso de la mano (b), vértices devueltos al origen (c), posición de los huesos al levantar el brazo (d) y vértices ubicados en su nueva posición (e)

Grados de libertad

Una vez definida la estructura jerárquica de huesos que componen el personaje, es necesario aplicar ciertas restricciones al mismo si se desea que el resultado sea siempre, y para todas las posibles animaciones que puedan generarse, anatómicamente coherente. Es aquí donde entran en juego las restricciones de grados de libertad de cada articulación.

El ser humano (y cualquier otro animal vertebrado) tiene ciertas limitaciones anatómicas a la hora de mover cada una de las partes articulables del cuerpo: el codo, por ejemplo, solo podrá doblarse con cierta torsión limitada, y sobre sí mismo y hacia la posición del hombro unos 80˚. Es necesario por tanto que esta información sea representada y transmitida de alguna forma al programa para que éste sepa qué huesos o articulaciones pueden animarse y hasta qué limites. Es lo que se llaman restricciones por grados de libertad.

En la figura 28, se ve el personaje de ejemplo, cuyo brazo se ha movido de forma exclusiva (el resto de huesos permanecen estáticos), primero doblándose hacia arriba y luego hacia abajo, algo que en la realidad resulta imposible sin aplicar cierta torsión al hueso del antebrazo o el hombro.

Figura 28: movimientos imposibles al no aplicar restricciones

Por tanto, cada hueso tendrá, para los tres posibles ejes de rotación, dos valores (máximo y mínimo) que representan el rango o intervalo de movimiento de la articulación, más allá del cual las posibles rotaciones se truncarán para evitar posiciones inconsistentes.

6.4.2. Formato del sistema óseo

El sistema óseo descrito, y cualquier otro sistema de deformación ósea, representa información específica para sólo cierto tipo de modelos (modelos deformables construidos de forma jerárquica) que no puede ser incluida en el formato .obj utilizado, según el estándar. Es necesario por tanto definir un formato de archivo que permita almacenar la información pertinente para construir la jerarquía ósea y asignarla a los vértices del modelo.

Se ha escogido un tipo de fichero propio (.sk, de *Skeleton*) para el almacenamiento de esta información. Se trata de un tipo de archivo binario (secuencia de bytes) que comienza con una cabecera de 4 bytes y un entero con el número de huesos definidos, para a continuación pasar a enumerar los datos correspondientes a los distintos huesos de la jerarquía, siendo el primero de ellos el considerado como hueso raíz.

Se enumera a continuación un extracto de este fichero (después de su interpretación como texto plano) a modo de ejemplo:

```
// CABECERA: SKELø_ (4 bytes)
```

Permite identificar el fichero como el formato definido, así como resaltar el inicio del mismo.

```
// Bone Count: 28 (1 byte)
```

Entero de 1 byte que indica el número de huesos descritos a continuación.

```
// name: Cadera (80 bytes)
```

Nombre del hueso actual; en realidad el nombre no tiene ningún uso ya que para referencias a uno u otro hueso se emplea el id del mismo, un entero que identifica su posición dentro del fichero.

```
// tx: 0.000000 (1 byte)
// ty: 8.012779 (1 byte)
// tz: 0.000000 (1 byte)
```

Datos de posición del hueso, relativos a la posición del hueso inmediatamente anterior en la jerarquía.

```
// rx: 0.000000 (1 byte)
// ry: 0.000000 (1 byte)
// rz: 0.000000 (1 byte)
```

Datos de rotación del hueso, relativos a la posición del hueso inmediatamente anterior en la jerarquía.

```
// max_rx: 0.000000 (1 byte)
// min_rx: 8.012779 (1 byte)
// max_ry: 0.000000 (1 byte)
// min_ry: 0.000000 (1 byte)
// max_ry: 0.000000 (1 byte)
// min_ry: 0.000000 (1 byte)
```

Describen las rotaciones máxima y mínima de los huesos en cada uno de los ejes de rotación (*x*, *y*, *z*), definiendo de este modo las restricciones de grados de libertad de cada articulación.

```
// parent:-1 (1 byte)
```

Entero que hace referencia al identificador del hueso padre dentro de la jerarquía (hueso inmediatamente anterior). Para el caso de huesos sin un hueso padre (p.ej. hueso raíz) el id será -1.

```
// childCount: 3 (1 byte)
```

Número de huesos que siguen de forma inmediata en la jerarquía al hueso actual (huesos hijos).

```
// child #0: 2  (1 byte)
// child #1: 7  (1 byte)
// child #2: 12 (1 byte)
```

Listado con los identificadores de los huesos “hijos” del hueso actual.

La carga e interpretación de este tipo de archivo se hace en la clase *CSkeleton* mediante el método *LoadSkeleton(),* el cual itera a lo largo del archivo leyendo (mediante sentencias *fread*) cada uno de los datos de forma secuencial.

6.5. Seguimiento del proyecto

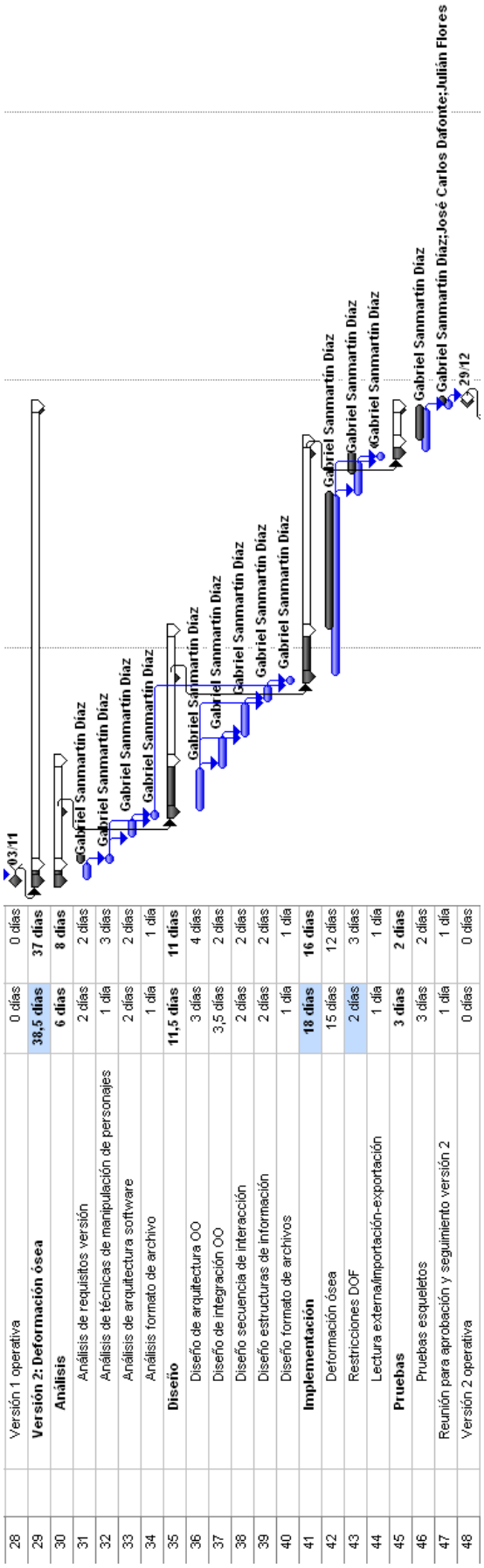
Partiendo del adelanto de dos días respecto a lo planificado resultante de la implementación del anterior incremento, el resultado total del seguimiento hasta este segundo incremento ha sido su finalización el mismo día que se había planificado, lo que significa que el desarrollo parcial de esta revisión se ha retrasado 2 días según la duración prevista.

Las causas de ello han sido que, pese a que la fase de análisis de técnicas de deformación de personajes ha durado tan sólo un día respecto a los tres que se preveían (pues finalmente el acuerdo es unánime en cuando a las técnicas a emplear, siendo la deformación ósea malla única la práctica habitual), el diseño ha sido más complejo de lo previsto debido a que su integración con el diseño de la iteración anterior ha requerido más trabajo del esperado para garantizar la consistencia y cohesión del comportamiento del sistema.

Asimismo, la complejidad para comprender el funcionamiento y, en especial, para conseguir resultados válidos en el desarrollo al hallarse involucrado complejos cálculos matriciales, ha conllevado un retraso importante de varios días en la implementación de esta revisión.

Por otro lado, las pruebas con varios modelos han supuesto el “skinning” de varios personajes, un proceso que por laborioso ha requerido más tiempo del deseable.

Como conclusión, la entrega de esta revisión se ha realizado el 29 de diciembre de 2009.



28	Versión 1 operativa	0 días	0 días
29	**Versión 2: Deformación ósea**	**38,5 días**	**37 días**
30	**Análisis**	**6 días**	**8 días**
31	Análisis de requisitos versión	2 días	2 días
32	Análisis de técnicas de manipulación de personajes	1 día	3 días
33	Análisis de arquitectura software	2 días	2 días
34	Análisis formato de archivo	1 día	1 día
35	**Diseño**	**11,5 días**	**11 días**
36	Diseño de arquitectura OO	3 días	4 días
37	Diseño de integración OO	3,5 días	2 días
38	Diseño secuencia de interacción	2 días	2 días
39	Diseño estructuras de información	2 días	2 días
40	Diseño formato de archivos	1 día	1 día
41	**Implementación**	**18 días**	**16 días**
42	Deformación ósea	15 días	12 días
43	Restricciones DOF	2 días	3 días
44	Lectura externa/importación-exportación	1 día	1 día
45	**Pruebas**	**3 días**	**2 días**
46	Pruebas esqueletos	3 días	2 días
47	Reunión para aprobación y seguimiento versión 2	1 día	1 día
48	Versión 2 operativa	0 días	0 días

Figura 29: Seguimiento del proyecto a la entrega de la versión 2

7. Versión 3: Cinemática Inversa

7.1. Objetivos

Es frecuente cuando se trabaja con aplicaciones de gráficos en tiempo real (y en especial en aquellas de carácter lúdico), hallarse en la necesidad de generar animaciones cuyos parámetros serán dependientes del estado actual del mundo virtual a la hora de dispararse el movimiento. En contraposición a éstas, se hallan las animaciones creadas con anterioridad y "grabadas" de alguna forma (almacenando las rotaciones, traslaciones y escalados de píxeles, vértices u objetos).

Por ejemplo, si un personaje debe recoger un número determinado de objetos a diferentes alturas, de no poder generarse de forma algorítmica estas animaciones, sería necesario hacer una animación por cada uno de estos objetos, lo cual puede resultar muy tedioso para un número elevado de éstos. Podemos complicar el ejemplo imaginando un objeto móvil que podrá ser alcanzado por el personaje en un momento arbitrario; se ve en este caso que resulta imposible crear una animación que alcance dicho objeto para todas las posibles posiciones del mismo.

En el caso de estudio, se plantea la necesidad de hacer un personaje elástico, de tal forma que alguna de sus extremidades se estire para alcanzar un punto arbitrario del mundo virtual. Pero antes de aplicarse dicho estiramiento, tiene sentido que el miembro en cuestión se oriente de forma animada hacia dicho objeto, para entonces comenzar a estirarse. Éste tipo de animaciones se generan mediante técnicas de cinemática inversa.

7.2. Análisis

7.2.1. Requisitos Funcionales

Código requisito	Nombre requisito	Descripción de requisito
RV03F01	Animaciones	Sobre el sistema óseo establecido se abre la posibilidad de animar las distintas extremidades mediante la aplicación de transformaciones básicas (rotación, traslación, escalado) sobre los huesos.
RV03F02	Sistema de cinemática inversa	Creación de un sistema que implemente una técnica para resolver problemas de cinemática inversa de forma similar al campo de la robótica.
RV03F03	Alcance de objetos	El sistema de cinemática inversa debe funcionar de tal forma que al hacer clic sobre un objeto de la escena, el sistema óseo asociado se acerque lo máximo posible a dicha posición (minimización).

Tabla 9: Requisitos funcionales de la versión 3

7.2.2. Requisitos no funcionales

Código requisito	Nombre requisito	Descripción de requisito
RV03NF01	Soluciones IK rápidas	Aun cuando la solución de cinemática inversa proporcionada no sea la más precisa, el cálculo de la misma debe ser lo más rápido posible, ya que ésta se hará en tiempo real al hacer clic sobre la escena e idealmente debe ser instantáneo.
RV03NF02	Solución de minimización	Para el cumplimiento del requisito RV03F03, la solución IK provista debe ofrecer un resultado concebible como "el mejor posible", en lugar de no hacer nada en caso de elegirse un punto fuera de alcance.
RV03NF03	Solución integrable con el sistema óseo definido	La solución de cinemática inversa proporcionada debe ser compatible con el modelo de deformación ósea implementado (y sus limitaciones DOF), u ofrecer los mecanismos o interfaces para ofrecer dicha compatibilidad.

Tabla 10: Requisitos no funcionales de la versión 3

7.2.3. Técnicas de IK

A la hora de efectuar el estudio de viabilidad y análisis de los diferentes métodos en el estado del arte para resolver problemas de cinemática inversa, resultó haber gran cantidad de ellos. Sin embargo, ninguno de los mismos es cercano a la perfección ni ofrece resultados óptimos para todas las situaciones, habiendo pues de perfilarse aun más el nivel de detalle en el análisis para no hallar soluciones eventualmente incorrectas. Entre los métodos analizados, se hallan los siguientes:

- IK analítica simple [24][14]
- Cyclic Coordinate Descent (CCD) [14][11]: Estructurada en árbol, Con límites desde la raíz, con límites de hijo a hijo, cambio de raíz dinámico, con 2 grados de libertad, 3 grados o 6 grados, solución secuencial de ángulos de Euler, solución numérica por cuaterniones…
- Solución por dinámica de cuerpos rígidos por coordenadas aumentadas o por coordenadas generalizadas.[14]
- Solución basada en partículas (SHAKE, RATTLE, Jakobsen) [14]
- Etc.

De entre todas estas técnicas, la conclusión general es que, a grandes rasgos, existen dos tipos diferenciados de soluciones para un sistema de cinemática inversa: soluciones cerradas o analíticas y soluciones numéricas.

Se usan las primeras para obtener cálculos exactos, mientras que las soluciones numéricas son adecuadas cuando el sistema se vuelve demasiado complicado para manejarlo con métodos cerrados.

En un primer intento de aproximación a la implementación de una solución de cinemática inversa, se escogió, por su precisión al seguir el modelo geométrico, la solución analítica. La causa es que esta representa la alternativa más intuitiva además de que, matemáticamente hablando, se trata de la más exacta. Además, su prueba en el ámbito de las dos dimensiones ofrecía resultados bastante positivos.

No obstante esta metodología se reveló pronto como inadecuada, por razones prácticas y de eficiencia, de tal modo que se optó por utilizar una solución basada en el método Cyclic Coordinate Descent con solución numérica por cuaterniones, que finalmente se mostró como la alternativa más adecuada. Además, la conversión a un entorno tridimensional de la solución análitica complicaba tanto los cálculos que estos se hacían directamente inabordables.

7.3. Diseño

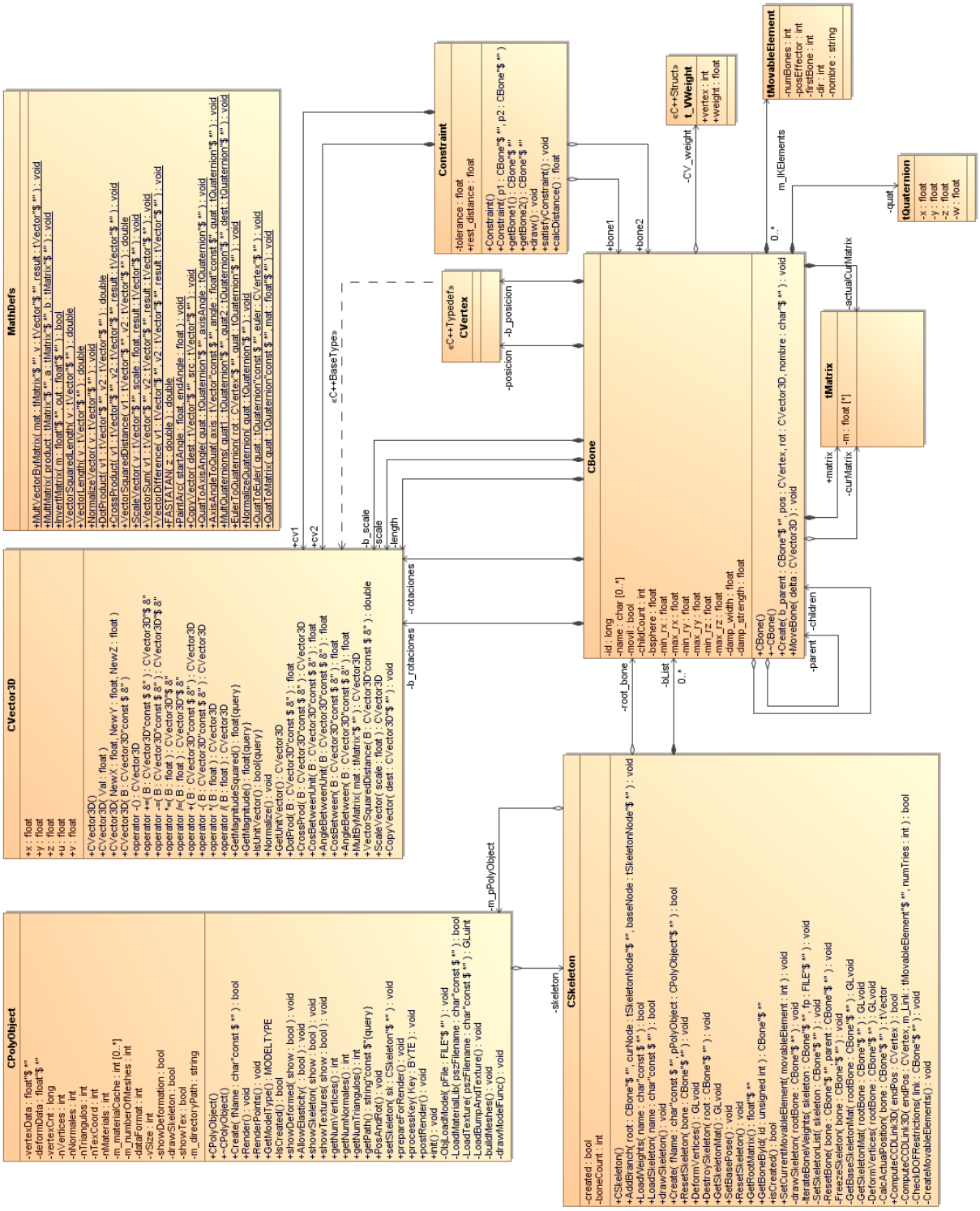


Figura 30: Diagrama de objetos para la versión 3

7.4. Implementación

7.4.1. Solución analítica

En un principio la solución escogida fue, como se ha comentado, la analítica simple, pues en el caso prototípico de un brazo 2D resulta relativamente fácil: simplemente implica invertir la trigonometría y cierta álgebra de vectores. Erróneamente se consideró que el paso a 3D sería trivial, sin embargo, al añadir esa tercera coordenada, se reveló que el sistema se volvía cada vez más complejo a medida que se lo dotaba de más flexibilidad, y que directamente resulta inviable para controlar el movimiento humano: una persona ofrece para cada articulación demasiados grados de libertad para los que resulta viable resolver este método. Incluso en el caso de un brazo en 2D, agregar el hombro como articulación supone un aumento demasiado grande de la complejidad [24].

A continuación se expone la solución implementada inicialmente:

Supóngase una cadena con dos eslabones, como se ve en la figura 31. La posición de P_2 viene dada por la rotación de los dos eslabones L_1 y L_2 de la cadena. Es posible resolver la **cinemática directa** del problema simplemente analizando cada uno de estos eslabones.

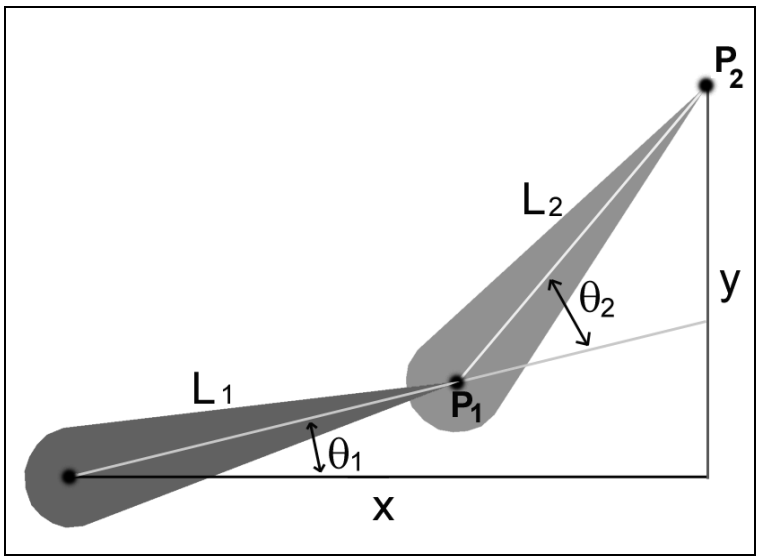


Figura 31: Representación de dos huesos

Las ecuaciones 14 y 15 presentan las soluciones analíticas de cinemática directa para P_1 y P_2.

$$P_{x_1} = L_1 * \cos(\theta_1)$$
$$P_{y_1} = L_1 * \sin(\theta_1) \quad (14)$$

$$P_{x_2} = P_{x_1} + (L_2 * \cos(\theta_1 + \theta_2))$$
$$P_{y_2} = P_{y_1} + (L_2 * \sin(\theta_1 + \theta_2)) \quad (15)$$

Sustituyendo en la ecuación 15 las expresiones de la ecuación 14, hallamos la relación de la expresión 16.

$$P_{x_2} = (L_1 * \cos(\theta_1)) + (L_2 * \cos(\theta_1 + \theta_2))$$
$$P_{y_2} = (L_1 * \sin(\theta_1)) + (L_2 * \sin(\theta_1 + \theta_2)) \quad (16)$$

Si se desean invertir estos cálculos, habrán de obtenerse los ángulos a partir de la posición final. Empezando por el ángulo 2 (θ_2) y dadas las identidades trigonométricas de las ecuaciones 17 y 18, obtenemos las expresiones de la ecuación 19.

$$\cos(a + b) = \cos(a) * \cos(b) - \sin(a) * \sin(b) \quad (17)$$
$$\sin(a + b) = \cos(a) * \sin(b) + \sin(a) * \cos(b) \quad (18)$$

$$\begin{aligned} P_{x_2} &= (L_1 * \cos(\theta_1)) + \big(L_2(\cos\theta_1 \cos\theta_2 - \sin\theta_1 \sin\theta_2)\big) \\ &= L_1 \cos\theta_1 + L_2 \cos\theta_1 \cos\theta_2 - L_2 \sin\theta_1 \sin\theta_2 \\ P_{y_2} &= (L_1 * \sin(\theta_1)) + \big(L_2(\cos\theta_1 \sin\theta_2 + \sin\theta_1 \cos\theta_2)\big) \\ &= L_1 \sin\theta_1 + L_2 \cos\theta_1 \sin\theta_2 + L_2 \sin\theta_1 \cos\theta_2 \end{aligned} \quad (19)$$

Elevando ambas partes al cuadrado:

$$\begin{aligned} {P_{x_2}}^2 &= {L_1}^2 \cos{\theta_1}^2 + {L_2}^2 \cos{\theta_1}^2 \cos{\theta_2}^2 - {L_2}^2 \sin{\theta_1}^2 \sin{\theta_2}^2 \\ &\quad + 2L_1L_2 \cos{\theta_1}^2 \cos\theta_2 - 2L_1L_2 \sin\theta_1 \sin\theta_2 \cos\theta_1 \\ &\quad - 2L_2 \sin\theta_1 \sin\theta_2 \cos\theta_1 \cos\theta_2 \\ {P_{y_2}}^2 &= {L_1}^2 \sin{\theta_1}^2 + {L_2}^2 \cos{\theta_1}^2 \sin{\theta_2}^2 + {L_2}^2 \sin{\theta_1}^2 \cos{\theta_2}^2 \\ &\quad + 2L_1L_2 \sin\theta_1 \cos\theta_1 \sin\theta_2 + 2L_1L_2 \sin{\theta_1}^2 \cos\theta_2 \\ &\quad + 2L_2 \sin\theta_1 \sin\theta_2 \cos\theta_1 \cos\theta_2 \end{aligned} \quad (20)$$

Si los sumamos:

$$\begin{aligned} {P_{x_2}}^2 + {P_{y_2}}^2 &= {L_1}^2 \cos{\theta_1}^2 + {L_1}^2 \sin{\theta_1}^2 + {L_2}^2 \cos{\theta_1}^2 \cos{\theta_2}^2 \\ &\quad + {L_2}^2 \cos{\theta_1}^2 \sin{\theta_2}^2 + {L_2}^2 \sin{\theta_1}^2 \sin{\theta_2}^2 \\ &\quad + {L_2}^2 \sin{\theta_1}^2 \cos{\theta_2}^2 + 2L_1L_2 \cos{\theta_1}^2 \cos\theta_2 \\ &\quad + 2L_1L_2 \sin{\theta_1}^2 \cos\theta_2 \end{aligned} \quad (21)$$

O lo que es lo mismo, obtenemos la serie de ecuaciones 22:

$$P_{x_2}{}^2 + P_{y_2}{}^2 = L_1{}^2(\cos\theta_1{}^2 + \sin\theta_1{}^2) + L_2{}^2\cos\theta_1{}^2(\cos\theta_2{}^2 + \sin\theta_2{}^2) + L_2{}^2\sin\theta_1{}^2(\sin\theta_2{}^2 + \cos\theta_2{}^2) + 2L_1L_2\cos\theta_2\ (\cos\theta_1{}^2 + \sin\theta_1{}^2)$$

$$P_{x_2}{}^2 + P_{y_2}{}^2 = L_1{}^2(1) + L_2{}^2\cos\theta_1{}^2(1) + L_2{}^2\sin\theta_1{}^2(1) + 2L_1L_2\cos\theta_2\ (1)$$

$$P_{x_2}{}^2 + P_{y_2}{}^2 = L_1{}^2 + L_2{}^2\cos\theta_1{}^2 + L_2{}^2\sin\theta_1{}^2 + 2L_1L_2\cos\theta_2$$

$$P_{x_2}{}^2 + P_{y_2}{}^2 = L_1{}^2 + L_2{}^2(\cos\theta_1{}^2 + \sin\theta_1{}^2) + 2L_1L_2\cos\theta_2$$

$$P_{x_2}{}^2 + P_{y_2}{}^2 = L_1{}^2 + +2L_1L_2\cos\theta_2 \qquad (22)$$

$$2L_1L_2\cos\theta_2 = P_{x_2}{}^2 + P_{y_2}{}^2 - L_1{}^2 - L_2{}^2$$

$$\cos\theta_2 = \frac{P_{x_2}{}^2 + P_{y_2}{}^2 - L_1{}^2 - L_2{}^2}{2L_1L_2}$$

$$\theta_2 = arccos\left(\frac{P_{x_2}{}^2 + P_{y_2}{}^2 - L_1{}^2 - L_2{}^2}{2L_1L_2}\right)$$

Ahora resolvamos para el ángulo 1. Según la imagen anterior:

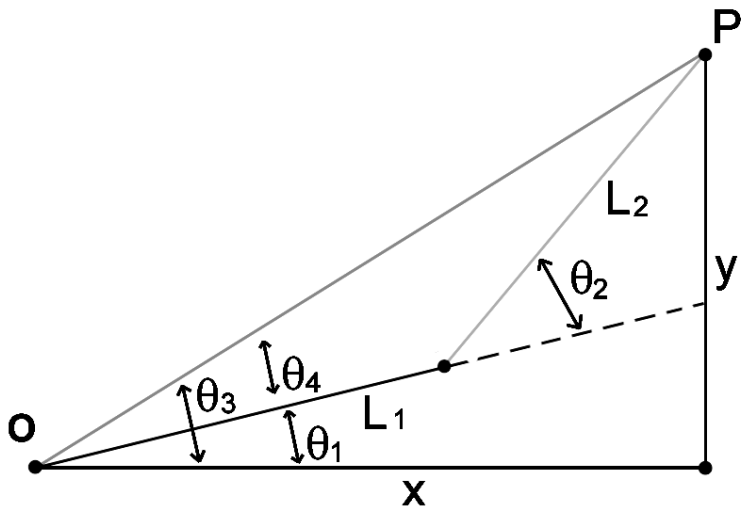


Figura 32: Cálculo de los ángulos de ambos huesos

De esta imagen pueden extraerse las relaciones 23-25:

$$\tan(\theta_3) = {}^{y}/_{x} \qquad (23)$$

$$\tan(\theta_4) = \frac{L_2\sin\theta_2}{L_2\cos\theta_2 + L_1} \qquad (24)$$

$$\theta_1 = \theta_3 - \theta_4 \tag{25}$$

Ahora, usando la identidad $\tan(a-b) = \frac{\tan(a)-\tan(b)}{1+\tan(a)\tan(b)}$, puede concluirse la expresión 26.

$$\tan(\theta_1) = \frac{\frac{y}{x} - \frac{L_2 \sin\theta_2}{L_2 \cos\theta_2 + L_1}}{1 + \left(\frac{y}{x} * \frac{L_2 \sin\theta_2}{L_2 \cos\theta_2 + L_1}\right)} \tag{26}$$

Si se multiplica por *x* y por el valor de $tan(\theta_4)$, se obtiene:

$$\tan(\theta_1) = \frac{y(L_2 \cos\theta_2 + L_1) - x(L_2 \sin\theta_2)}{x(L_2 \cos\theta_2 + L_1) + y(L_2 \sin\theta_2)} \tag{27}$$

$$\theta_1 = arctan\left(\frac{y(L_2 \cos\theta_2 + L_1) - x(L_2 \sin\theta_2)}{x(L_2 \cos\theta_2 + L_1) + y(L_2 \sin\theta_2)}\right) \tag{28}$$

Y eso es todo, se ha resuelto este problema IK. Para cualquier posición (*x*, *y*) es posible calcular los ángulos necesarios para alcanzarlo.

Esta solución se ha puesto en práctica y puede verse el resultado en la secuencia de imágenes de la figura 33 (nótese la limitación de su uso al plano *XY*).

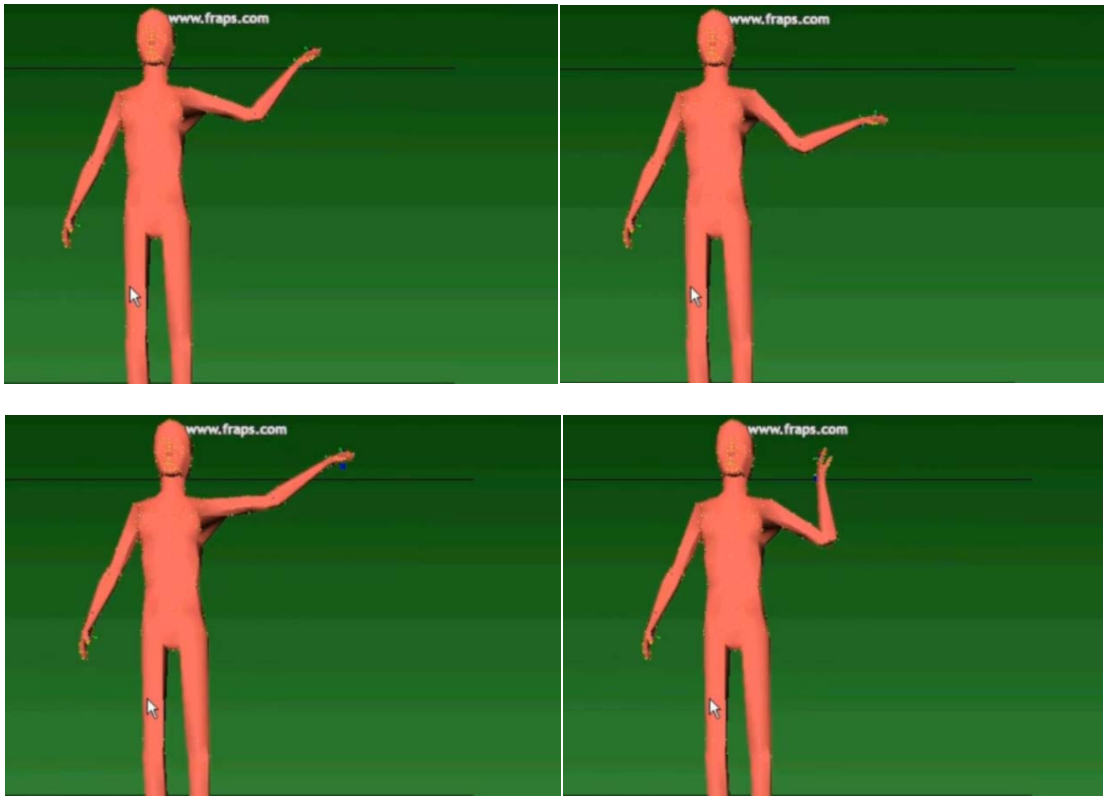


Figura 33: Secuencia de una solución analítica de IK

Esta técnica es la que se conoce como solución cerrada o analítica. Tiene la ventaja de ser una solución exacta y su cálculo es muy rápido. También permite saber de forma rápida y sencilla si un punto es o no alcanzable por el efector. Sin embargo, si no lo es, esta técnica ni siquiera

intentará acercarse, sólo informará de la posibilidad de ofrecer una solución. Esto no es conveniente para el caso de estudio, ya que precisamente los puntos a los que se deseará que los elementos estirables (p. ej. brazos) se desplacen serán puntos fuera de alcance, pudiendo llegar a ellos únicamente tras aplicarse el comportamiento elástico (fig. 34).

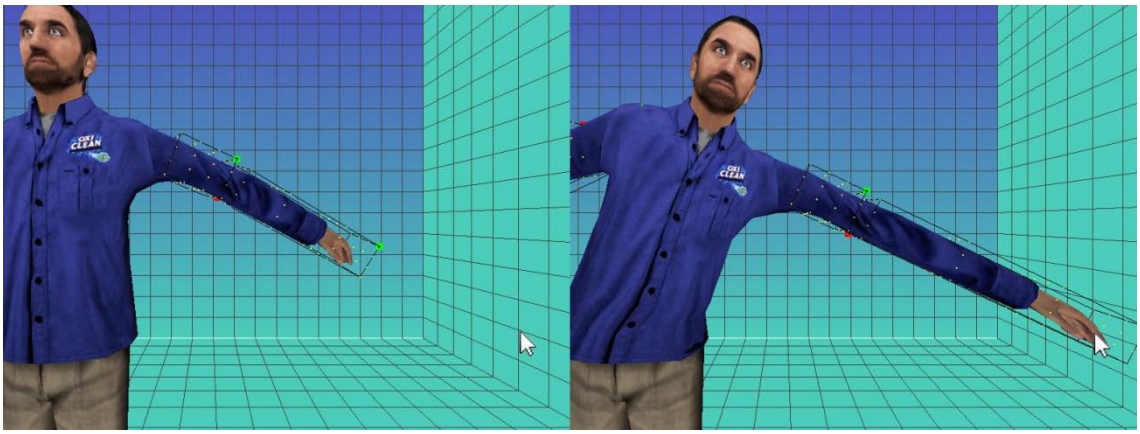


Figura 34: Personaje estirándose para alcanzar el puntero

Otra desventaja ya enunciada con anterioridad de esta solución es que se vuelve cada vez más compleja de resolver a medida que se aumentan los grados de libertad. De hecho, sólo aumentar un grado más de libertad dispara ostensiblemente la cantidad de cálculos que necesitamos resolver para obtener una solución. Dado que el sistema planteado (un personaje humano con grados de libertad realistas) requiere atravesar una jerarquía de más de un eslabón (hombro, antebrazo, brazo, muñeca, por ejemplo) una solución cerrada como ésta simplemente deja de ser matemáticamente viable.

7.4.2. Cyclic-Coordinate Descent

Los problemas más complejos requieren, por lo tanto, de una aproximación iterativa. Por suerte, se pueden importar ciertas ideas del mundo de la robótica. Este campo ha establecido diferentes métodos para la resolución de cinemática inversa en un sistema arbitrario. Estas soluciones se basan generalmente en técnicas de inversión de matrices con ciertas optimizaciones. La inversión de matrices es un proceso complicado que es computacionalmente demasiado costoso y padece además problemas derivados de la inestabilidad numérica.

Pero la cinemática inversa es, en su base, un problema de minimización del error [9]. En el caso de un brazo articulado que debe alcanzar cierta posición, una forma de ver el problema podría ser la de intentar minimizar la distancia entre el punto objetivo y el efector final de la cadena, por ejemplo, la mano. Esto puede conseguirse mediante el ajuste de los ángulos de las articulaciones de tal forma que se minimice dicha distancia.

La técnica Cyclic-Coordiante Descent fue descrita por primera vez en [39], y se trata de un método de optimización que trata de minimizar el error del sistema ajustando el ángulo de cada articulación por separado. Comienza por el último eslabón de la cadena (efector) y continúa hacia atrás en la jerarquía, ajustando los ángulos de cada uno en el proceso.

Comenzando por el último enlace en la cadena, se crea primero un vector desde la raíz del eslabón actual, *R*, a la posición actual del efector, *E* (ver figura 35). A continuación se calcula otro vector desde *R* a la posición deseada, *D*. Lo que se desea por tanto es determinar el ángulo α necesario para rotar el vector *RE* para que sea paralelo con en el vector *RD*.

El producto escalar entre dos vectores se define como la ecuación 27.

$$\cos(\propto) = \overrightarrow{RD} \cdot \overrightarrow{RE} \qquad (27)$$

Tomando la inversa del coseno del producto vectorial, se puede calcular el ángulo entre ambos vectores.

Sin embargo, dado que el producto escalar sólo proporciona el ángulo, es necesario también conocer la dirección en la que se debe rotar *R*, para lo cual se recurre al producto vectorial. Éste devuelve un vector perpendicular a ambos vectores. Simplemente comprobando el signo de la componente *Z* de este vector, podemos conocer en qué dirección se debe rotar el elemento actual de la cadena.

Se repite ahora el proceso en el elemento anterior de la cadena, prosiguiendo de la misma forma hasta alcanzar el elemento raíz, para recomenzar de nuevo en el efector.

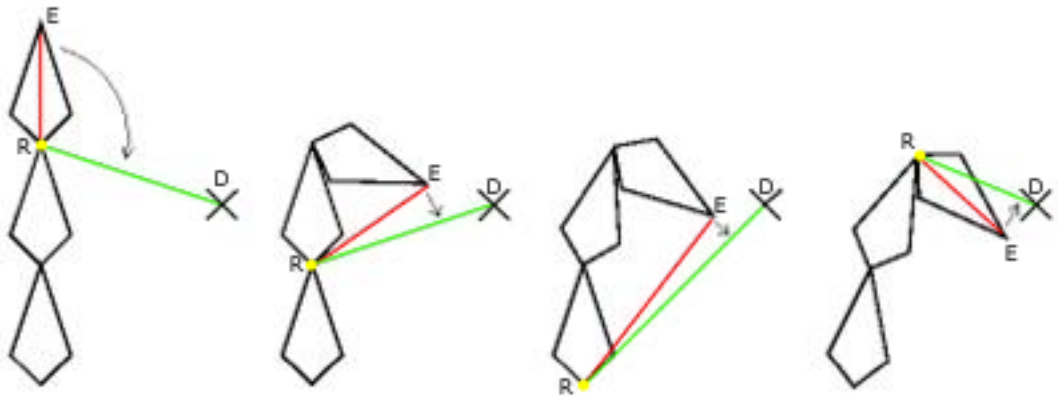


Figura 35: Aplicación eslabón a eslabón del algoritmo CCD

Este proceso se reitera entonces hasta que se alcanza una solución lo suficientemente cercana a la posición deseada como para resultar aceptable, o bien hasta que se ha repetido un número máximo de veces. Esta última condición es necesaria para aquellas posiciones que son simplemente inalcanzables por el sistema, como condición de parada.

7.4.3. El paso a las 3D

El proceso de actualización de las rotaciones de cada hueso, hasta ahora, pasaba por actualizar para cada hueso sus atributos *rot.x*, *rot.y*, *rot.z*, para luego aplicar en la representación del personaje cada una de estas rotaciones individualmente mediante la primitiva de OpenGL *glRotatef()* en la función *getSkeletonMat()* de *CSkeleton*, fijando de forma secuencial cada uno de los ángulos de rotación, en el orden *X, Y, Z*. Esto es lo que se conoce como rotación a través de matrices o ángulos de Euler.

Pero este método, aunque útil para la gran mayoría de las situaciones y muy intuitivo, presenta un problema muy conocido para el mundo de la animación y que trae de cabeza a los profesionales del mismo: el problema del "Gimbal Lock".

Gimbal Lock

El "Gimbal Lock" o "bloqueo de cardán" es un fenómeno que ocurre cuando dos ejes rotacionales de un objeto apuntan en la misma dirección. El resultado es que la rotación del objeto no produce los resultados esperados. Este problema es una limitación característica de matrices rotacionales (o vectores rotacionales) basadas en ángulos de Euler.

La razón de este problema es que usando ángulos de Euler se evalúa cada eje de forma independiente, en un orden predeterminado (generalmente *X*, *Y*, *Z*). Esto significa que primero los objetos se mueven primero a través del eje *X*, cuando esa operación se ha completado se pasa al eje *Y*, y por último se termina con el eje *Z*.

El problema del "Gimbal Lock" ocurre cuando, por ejemplo, rotamos en el eje *Y* 90 grados. Dado que la componente *X* ya ha sido evaluada no se ve afectada por los otros dos ejes. El resultado es que los ejes *X* y *Z* se alinean alrededor del mismo eje, como puede verse en la figura 36b, y por tanto se pierde un grado de libertad de rotación (rotar en *X* y *Z* produce el mismo resultado).

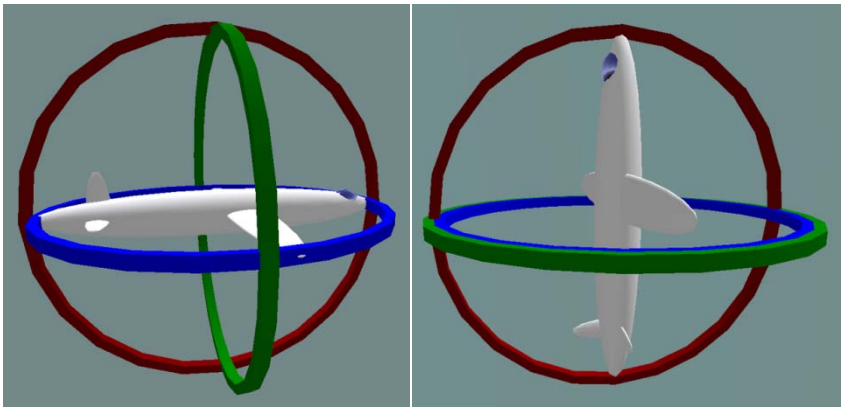


Figura 36: Gimbal Lock (dcha.)

La solución a este problema, presente en las soluciones CCD cuando se utilizan las tres dimensiones de rotación (en lugar de las 2 que veníamos usando), pasa por utilizar un sistema diferente para aplicar las rotaciones: los cuaterniones.

Cuaterniones

Los cuaterniones [15] son un sistema numérico que extiende los números complejos. Fueron descritos por primera vez por el matemático irlandés William Rowan Hamilton en 1843 y se aplican a la mecánica en espacios tridimensionales. Un cuaternión viene descrito por la ecuación 28.

$$q = a + bi + cj + dk \qquad (28)$$

Básicamente, puede representarse un cuaternión a partir de un vector (*x,y,z*) y un escalar, *w*.

La rotación por cuaterniones es más potente y robusta que a través de ángulos de Euler: en este caso, se evalúan los tres ejes de rotación al mismo tiempo para hallar la dirección de desplazamiento y un cuarto valor (la componente *w* de un cuaternión) para indicar a la matriz cuánto desplazarse. La ventaja de usar esta técnica es que no hay que preocuparse por el "Gimbal Lock": es directamente imposible que se produzca dado que los tres ejes se actualizan simultáneamente. La desventaja es que son mucho más complicados de interpretar por parte del programador, y la descripción de un sistema de ángulos a través de cuaterniones no es intuitiva; por ello los ángulos máximo y mínimo de las restricciones de grados de libertad de cada hueso se especifican como ángulos de Euler en lugar de en cuaterniones.

En el método CCD descrito, se utiliza el producto escalar para calcular el ángulo necesario para rotar la articulación de tal manera que se alcance el punto deseado, y el producto vectorial para determinar el sentido de rotación. Lo que realmente proporciona el producto vectorial es el **eje** de rotación, que será un eje arbitrario en 3D. Convirtiendo este eje de rotación y el ángulo de rotación a un cuaternión (ver más abajo algoritmo AxisAngleToQuat), se obtendrá todo lo necesario para aplicar en la función *getSkeletonMat()* la rotación requerida, pero esta vez en lugar de usar sucesivas llamadas de *glRotatef()*, para los distintos ejes (*x,y,z)* de rotación, se utilizará una única llamada con todas las rotaciones de forma simultánea.

```
void MathDefs::AxisAngleToQuat(const tVector *axis, const float
angle, tQuaternion *quat)
{
      float sin_a;
      float cos_a;
      sin_a = sin (angle/2);
      cos_a = cos (angle/2);

      quat->w = cos_a;
      quat->x = axis->x * sin_a;
      quat->y = axis->y * sin_a;
      quat->z = axis->z * sin_a;
}
```

Para combinar las rotaciones de dos cuaterniones, basta multiplicarlos entre sí, de modo que después de obtenida la rotación calculada en la iteración CCD, se multiplicará por el cuaternión asignado al hueso actual (un atributo dentro de su clase), para obtener la rotación final del mismo.

7.4.4. Gestión de DOF

Para poder imponer límites en los grados de libertad de cada articulación como se había hecho en la versión anterior, y que éstos se respeten en las soluciones de cinemática inversa calculadas, será necesario hacer la comprobación de que así es después de cada iteración del método. Por suerte, aplicar la verificación de estas restricciones es muy sencillo con el método CCD. Debido a que esta técnica trabaja de forma individual con cada articulación, es fácil imponer limitaciones: cuando el método vaya a actualizar la rotación en la articulación, se comprueba si el valor de rotación cae fuera de los límites, y en caso positivo, simplemente se trunca su valor al límite rebasado. El resto de los huesos de la cadena involucrada en el movimiento se utilizarán para alcanzar la solución en sucesivas iteraciones.

Sin embargo, al utilizar cuaterniones entra en juego un problema: en el formato .sk se han definido las restricciones de grados de libertad como ángulos de Euler máximos y mínimos para cada uno de los 3 ejes, pero ahora se está trabajando con cuaterniones, así que se hace imperativa una nueva conversión para verificar, para cada solución CCD, que se están cumpliendo las restricciones anatómicas impuestas.

Posteriormente, debe convertirse de nuevo de ángulos de Euler (con las correcciones ya aplicadas) a cuaterniones para que se actualicen los posibles cambios. Esto ocupa ciclos de computación pero no existe otra forma de permanecer dentro de una definición intuitiva del esqueleto del personaje y a la vez trabajar en el espacio de cuaterniones.

7.5. Seguimiento del proyecto

El desarrollo de este incremento, en contraposición a los anteriores, sí ha conllevado un retraso más que considerable en la visión global del desarrollo del proyecto. Las causas se han hallado básicamente en la gran cantidad de técnicas existentes para la aplicación de cinemática inversa, lo que ha llevado a un estudio prolongado de las mismas para hallar la más ajustada, retrasando la fase de análisis.

Además, la elección errónea en primera instancia del sistema analítico para la resolución de problemas de cinemática inversa ha comportado un nuevo retraso en la implementación, que ha alargado en más de 5 días la etapa de programación de esta revisión.

Afortunadamente las pruebas han permitido reducir esta prolongación gracias a que la estimación de las mismas era demasiado conservadora y finalmente pudieron agilizarse de manera considerable.

Como resultado, la entrega de la versión 3 se ha realizado el 9 del marzo de 2010, lo que supone un importante retraso de algo más de una semana respecto a la fecha fijada como línea de base.

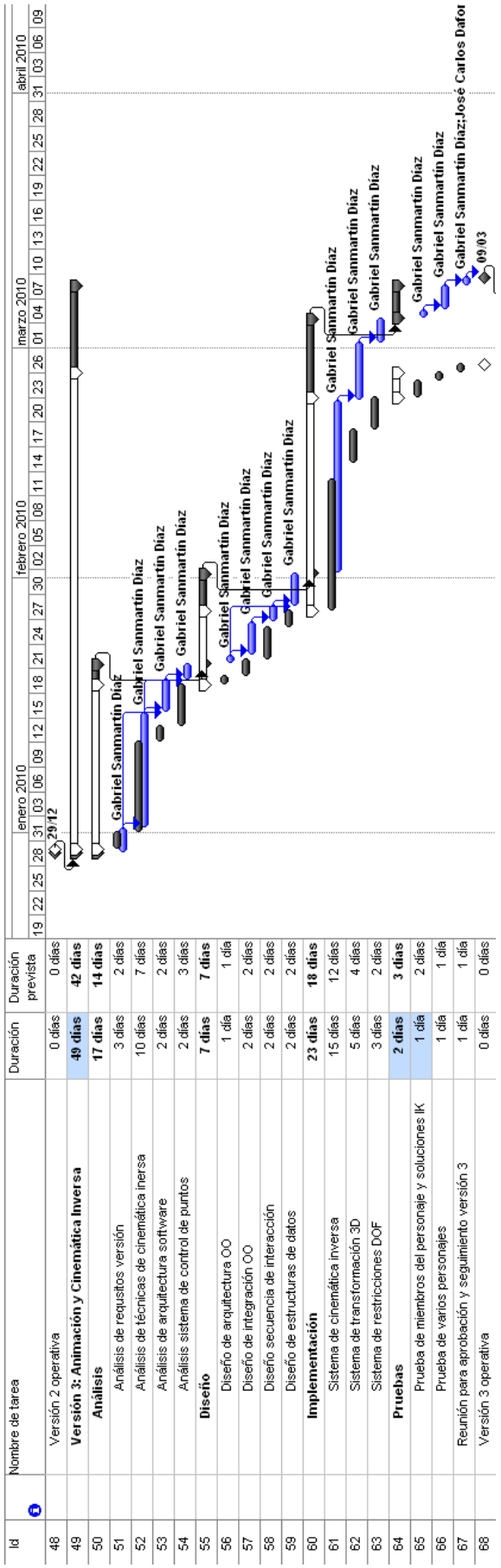


Figura 37: Seguimiento del proyecto a la entrega de la versión 3

8. Versión 4: Elasticidad

8.1. Objetivos

Se proporcionará un sistema para dotar de comportamiento elástico a objetos 3D, dirigido a personajes animados, para su uso en entornos "cartoon" no reales o lúdicos que mantenga una coherencia visual con las propiedades físicas y estéticas del entorno.

La deformación permitirá el trabajo de forma localizada en ciertas áreas del objeto afectado, pudiéndose utilizar un número arbitrario de deformaciones y organizarse éstas en una estructura jerárquica controlable.

La idea es poder controlar esta deformación de forma transparente al usuario, idealmente con el ratón o el teclado, aplicándose los parámetros elásticos y su comportamiento al liberarse la fuerza de estiramiento aplicada sobre el objeto.

8.2. Análisis

8.2.1. Requisitos Funcionales

Código requisito	Nombre requisito	Descripción de requisito
RV04F01	Crear un sistema sencillo y flexible de deformación de píxeles	Se proporcionará una interfaz para el manejo o creación de sistemas de deformación a nivel de píxel (más allá de rotación, traslación, escaldos globales) sobre los modelos cargados en la escena.
RV04F02	Dotar de elasticidad a ese sistema	Este sistema de deformación será manipulado de tal forma que al aplicarse una deformación el comportamiento resultante sea de carácter elástico, tal y como se comportaría una goma elástica.
RV04F03	Proveer de un sistema de almacenamiento externo de la información	Las deformaciones definidas por el usuario deben poder almacenarse de forma externa para su uso entre varias ejecuciones.

Tabla 11: Requisitos funcionales de la versión 4

8.2.2. Requisitos no funcionales

Código requisito	Nombre requisito	Descripción de requisito
RV04NF01	Deformaciones locales	Las deformaciones de píxeles deben poder aplicarse de forma local, para evitar el resultado "gelatinoso", y para elastizar sólo ciertas partes de los objetos.
RV04NF02	Deformación intuitivamente consistente	El comportamiento (elástico o no) de todo el sistema de deformaciones debe ofrecer unos resultados visuales intuitivamente consistentes, que aun siendo imposibles de

	hallar en el mundo real, resulten verosímiles.

Tabla 12: Requisitos no funcionales de la versión 4

8.2.3. Sistemas de muelles y masas

Deformar los vértices dentro de una malla entra dentro del dominio de los cuerpos blandos. Al abordar el estudio de la deformación de esta clase de objetos (para la que existen múltiples técnicas), una de las alternativas que se plantearon a la hora de poner en práctica dicho requerimiento fue el empleo de algoritmos basados en sistemas de muelles y masas, más conocidos por su nombre en inglés "mass-spring systems"; una técnica frecuentemente utilizada en la simulación de ropas [4] o, en menor medida, en la simulación de física para cuerpos blandos o cuerpos deformables [34].

El modelo de telas

Un sistema de muelles y masas consiste en concebir cada uno de los vértices del modelo como una partícula con masa, conectada con sus partículas vecinas a través de muelles (puede pensarse en estos muelles como las fibras de una prenda o tela), los cuales ejercerán una fuerza elástica al desplazarse cualesquiera vértices de su posición original, tal y como lo haría un muelle elástico.

La idea pasa por modelar un cuerpo blando utilizando esta técnica, y no un simple modelado de telas; para ello se define una tipología de muelles completa, distinguiendo varios tipos de muelles; el sistema escogido definía tres clases diferentes [29][5] a saber:

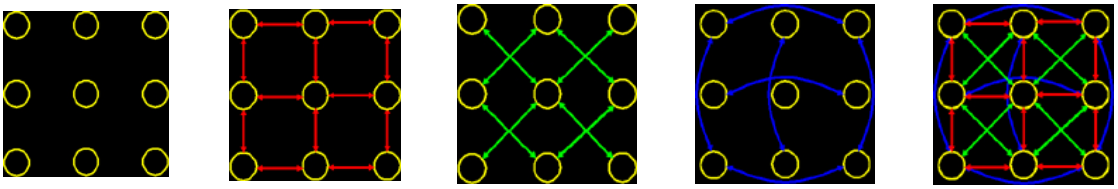


Figura 38: tipos de muelle, estructurales (b), de corte (c), de flexión (d), y todos (e)

- Muelles estructurales: controlan la tracción de la tela y la tensión de compresión. Son conexiones verticales y horizontales entre partículas (fig. 38b)
- Muelles de corte o cizalladura: controlan el estrés de cizalladura. Se trata de conexiones directas diagonales. (fig. 38c)
- Muelles de flexión: controlan las dobleces de la tela. Conectan vértices intercalados tanto vertical como horizontalmente. (fig. 38d)

Una vez conectados los muelles, es necesaria una forma de calcular la acción de una fuerza sobre cada partícula a lo largo del tiempo, en términos de cómo afecta a su posición. Para ello, se hace uso de la segunda ley de Newton para obtener la aceleración, y a partir de ésta, se obtiene la posición a través de una integración numérica, dado que la posición no es más que la segunda derivada de la aceleración.

En otras palabras, dada la partícula o masa *i*, *j* en una retícula de *m*n* masas, se calcula su posición como indican las ecuaciones 29-31:

$$X_{i,j}(t+\Delta t) = X_{i,j}(t) + \partial V_{i,j}(t+\Delta t) \quad (29)$$

$$V_{i.j}(t+\Delta t) = V_{i,j}(t) + \partial a_{i,j}(t+\Delta t) \quad (30)$$

$$a_{i,j}(t+\Delta t) = {}^{1}/_{m} * F_{i,j}(t) \quad (31)$$

Sin embargo, pronto se comprobó en los resultados que esta técnica no era la más indicada para el objetivo del caso propuesto: por un lado el resultado era muy diferente a lo deseado, con un comportamiento más cercano al de una prenda de ropa elástica que al de un cuerpo sólido con tales propiedades, proporcionando movimientos demasiado restringidos en distancia y una vuelta al estado de reposo mucho más inestable y oscilante de lo esperable. Por otro lado, los resultados de eficiencia hacían imposible construir una retícula de muelles y masas para un modelo de complejidad media-alta como los utilizados en el desarrollo; ofreciendo para una retícula de 50x50 partículas y muelles estructurales, de corte y de flexión, resultados muy pobres en cuando a rendimiento.

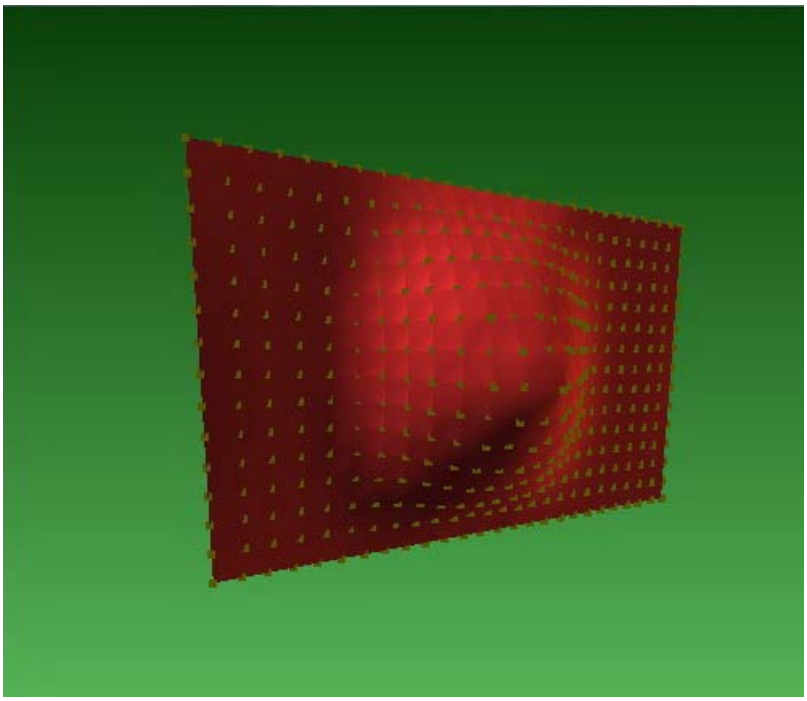


Figura 39: Resultado de la implementación de un sistema de muelles y masas

8.2.4. Free-Form Deformation

Una vez desechado el método de muelles y masas, dentro de las múltiples metodologías de deformación a nivel de vértice, una que resultó ser particularmente apropiada para la propuesta fue la técnica de Free-Form Deformation (en adelante, FFD), introducida en 1986 por Sederburg & Parry [37], la cual es además ampliamente conocida.

Free-Form Deformation consiste en la transformación de un objeto mediante su introducción en un sistema de coordenadas ad-hoc, que se deformará de tal manera que sus ejes inicialmente rectos pasen a ser curvos, implícitamente arrastrando esa deformación a los

objetos dentro de él [7]. Se almacenan las nuevas posiciones de los píxeles tras dicha deformación para transformarlas de nuevo al sistema de coordenadas regular. Ya en [37] se propone la analogía con un paralelepípedo flexible de plástico que envuelve el objeto, también flexible, que se deforma a medida que lo hace el plástico que lo envuelve.

Básicamente, la FFD es la transformación de un objeto desde su apariencia tal y como fue modelado (situación de reposo) a su estado deformado, lo que se consigue mediante la deformación del sistema de coordenadas del objeto en los siguientes pasos:

1. El objeto a deformar se introduce en un sistema de coordenadas regular definido por tres ejes mutuamente perpendiculares, o los clásicos *X*,*Y*,*Z*.
2. El sistema de coordenadas se deforma, permitiendo que los ejes anteriormente rectos se conviertan en curvos. Áreas dentro del sistema de coordenadas pueden contraerse o expandirse.
3. Las posiciones de los vértices de los objetos en el anterior sistema de coordenadas se actualizan para situarse en el punto en el que terminaron tras deformarse el sistema de coordenadas.

Este procedimiento dará como resultado un objeto deformado de tal manera que resulte "intuitivamente consistente" con cómo se ha deformado el sistema de coordenadas, es decir, que el objeto se verá de la forma que se espera que lo haga.

Objetos y sistemas de coordenadas

Habitualmente, un objeto constituido por una malla de polígonos tiene una forma que viene definida por las coordenadas de sus vértices en el sistema local de coordenadas. La piedra angular de FFD es que la posición de los vértices del objeto no se modifica directamente; en su lugar, el sistema local de coordenadas del objeto es lo que se modifica, lo que cambia implícitamente la posición de los vértices del objeto en el espacio.

FFD tiene, por tanto, la ventaja de ser una técnica muy general. Cualquier efecto creado mediante FFD se puede aplicar sobre cualquier objeto con resultados similares, sin limitación por el tipo de objeto: es válido tanto para objetos poligonales, curvas de Bézier, "B-Splines" o cualquier otra técnica, lo que lo convierte en ideal para el trabajo con gráficos 3D: No se modifica directamente la posición de los vértices del objeto, en su lugar, se altera el sistema local de coordenadas en el que dicho objeto se introduce, con lo que implícitamente se modifica la posición en el espacio de sus vértices.

En lugar de pensar en términos de "este objeto está rotando", es más útil concebirlo pensando en sistemas de coordenadas: "el sistema local de coordenadas del objeto está rotando, de tal forma que el objeto parece que rota". Ésta es la forma en que trabajan APIs gráficas como la utilizada, OpenGL.

Pero las operaciones como rotación o traslación son sencillas de realizar y de concebir, pues aplican exactamente la misma operación a cada vértice del objeto: las líneas rectas del objeto antes de la operación siguen siéndolo tras ella. Si se concibe un sistema de coordenadas que permita doblarse o estirarse de forma directa, el objeto se comportará como deseamos.

Para conseguir esto se debe definir un sistema de coordenadas con mayor libertad que el que todos conocemos. Este sistema se basa en volúmenes de Bézier, que son una extensión de las curvas y superficies de Bézier.

Volúmenes de Bézier

La base de la técnica de FFD se fundamenta en los volúmenes de Bézier, que no son más que una generalización de las conocidas curvas de Bézier. Para una explicación detallada de éstas, remítase a la introducción de esta memoria.

Baste decir que una curva de Bézier viene definida por la ecuación paramétrica *Q(u)*, que tiene la forma de la ecuación 32-33.

$$Q(u) = \sum_{i=0}^{x} \boldsymbol{P_i} B_{i,x}(u)\,, \qquad u\,\epsilon[0,1] \tag{32}$$

$$B_{i,x}(u) = \binom{x}{i} e^i (1-u)^{x-1}, \qquad i = 0 \ldots n \tag{33}$$

Para grado 3, la ecuación puede expandirse de tal modo que se obtienen cuatro ecuaciones que dependen únicamente del parámetro *u*. Para evaluar la posición en una curva de Bézier cúbica para cualquier valor del parámetro *u*:

$$Q(u) = \sum_{i=0}^{3} P_i B_{i,3}(u) = P_0 B_{0,3}(u) + P_1 B_{1,3}(u) + P_2 B_{2,3}(u) + P_3 B_{3,3}(u) \tag{34}$$

Calculando el valor de la función para un intervalo fijo y conectando los puntos resultantes con líneas rectas se puede dibujar una curva de Bézier.

Extender esta teoría a volúmenes simplemente supone aumentar el número de puntos de control y añadir otros dos parámetros a la ecuación de Bézier. El conjunto de puntos de control *P* ahora contiene 64 vectores en una matriz 4x4x4, y la curva vendrá definida en términos de tres parámetros, como se muestra en la ecuación 35.

$$Q(u,v,w) = \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} P_{i,j,k} B_{i,3}(u) B_{j,3}(v) B_{k,3}(w) \tag{35}$$

Para poder envolver un objeto dentro de uno de estos volúmenes, lo más sencillo pasa por construir una forma cúbica basada en un volumen de Bézier, disponiendo los puntos en una retícula. En este caso los puntos de control, dibujados en un espacio 3D, tendrían el aspecto de la fig. 40.

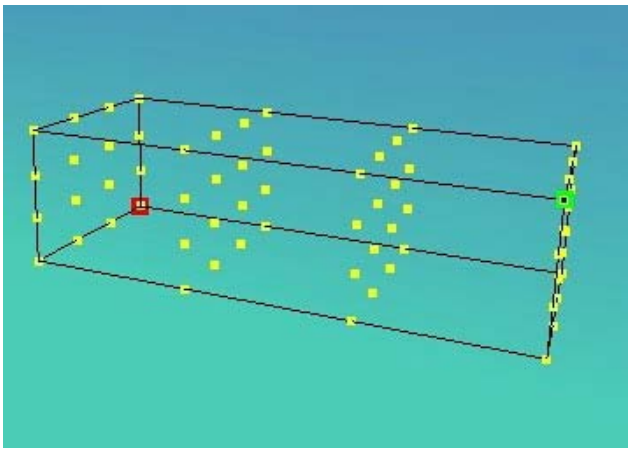


Figura 40: Retícula de deformación

Si movemos uno de estos puntos $P_{i,j,k}$ los ejes de coordenadas dejarán de ser rectos y pasarán a ser curvos.

Detalles de cómo llevar a cabo estas transformaciones se abordan en la implementación.

8.3. Diseño

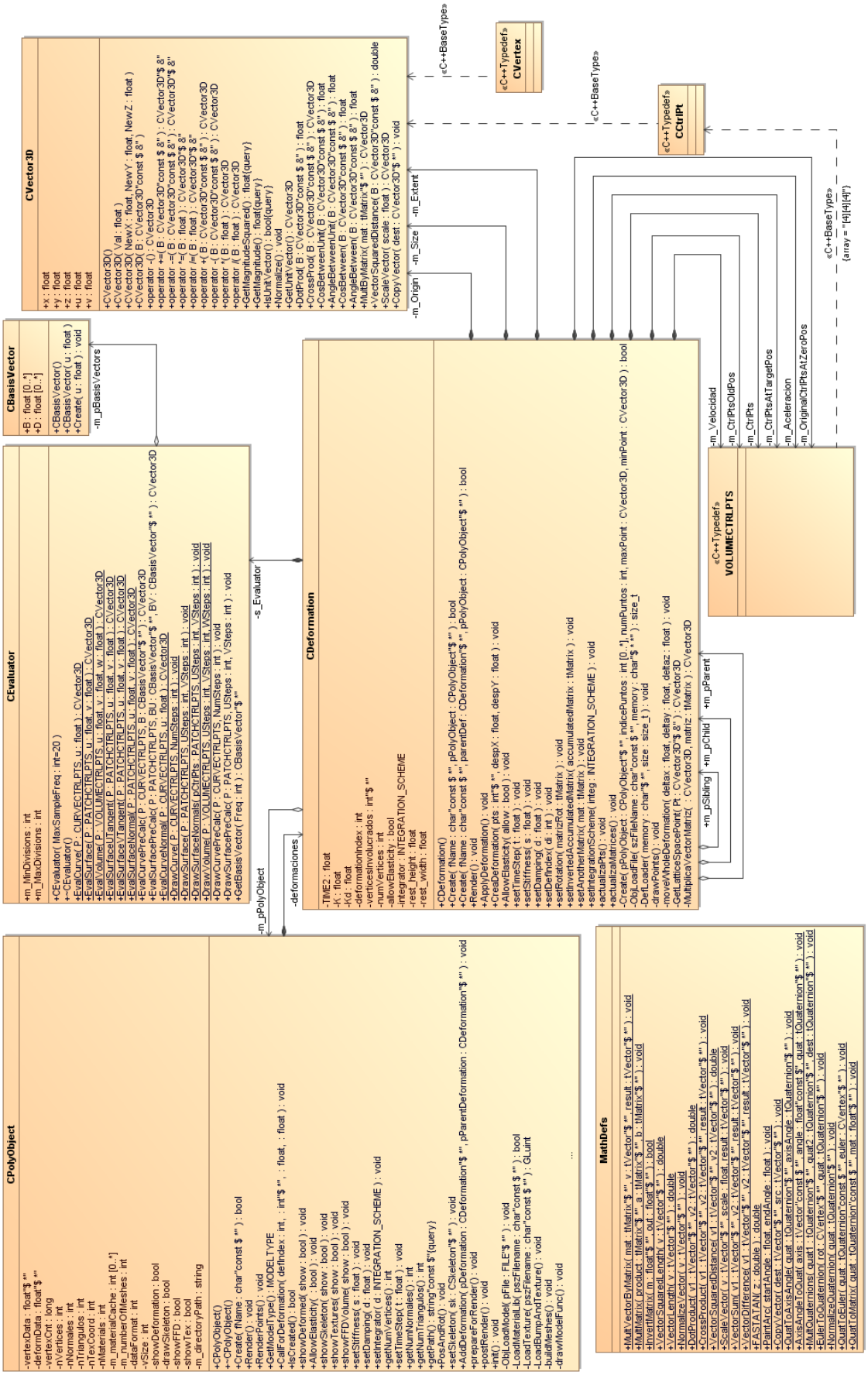


Figura 41: Diagrama de objetos para la versión 4

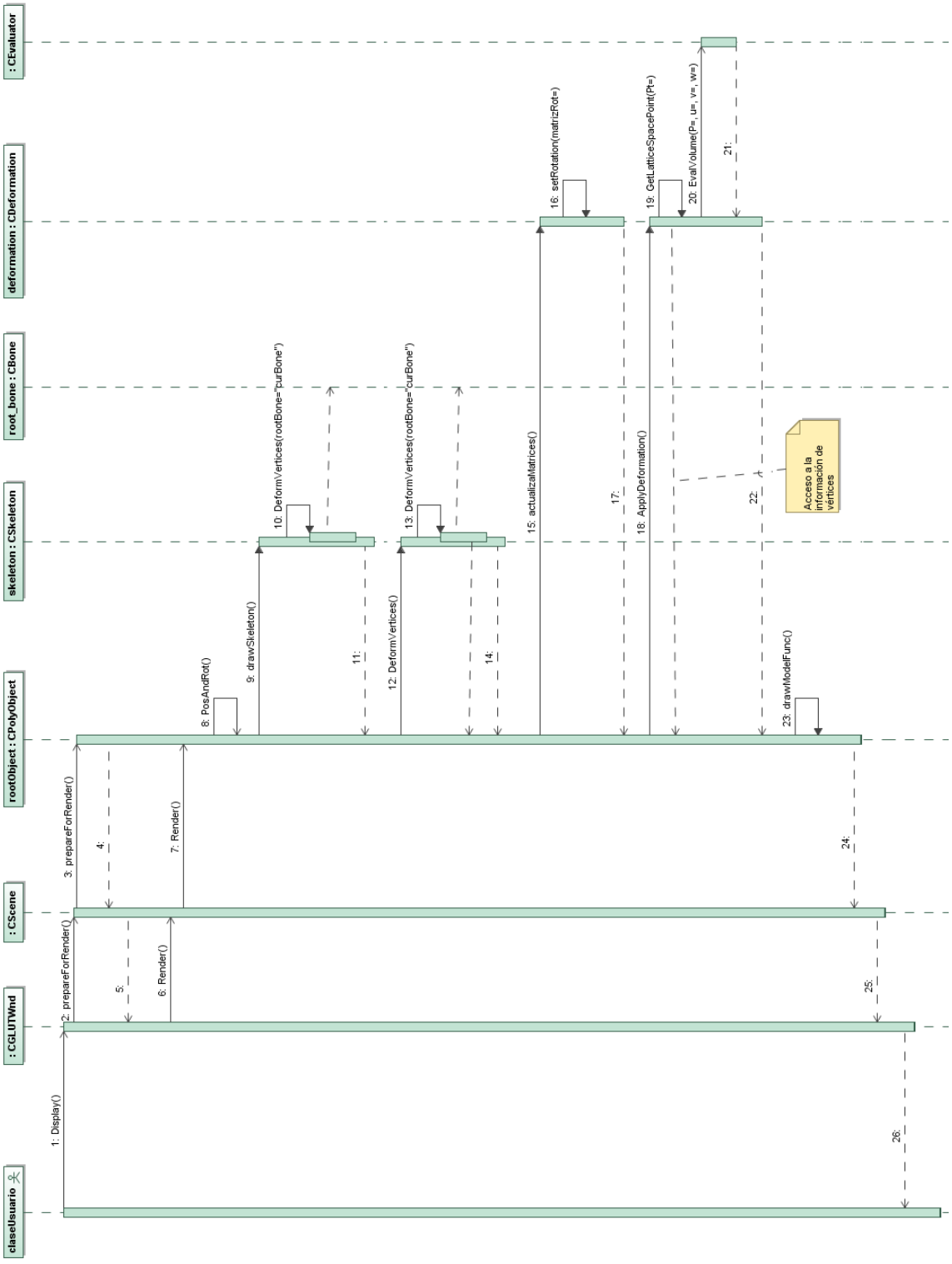


Figura 42: Diagrama de secuencia del caso de uso renderizar escena, versión 4

8.4. Implementación

8.4.1. Free-Form Deformation

Para transformar cualquier punto en el sistema de coordenadas definido por la retícula del volumen de Bézier, se debe considerar que cualquier posición se representará en función de tres coordenadas, *u*, *v* y *w*, que corresponderán con los parámetros del volumen de Bézier. Así, cualquier punto arbitrario x tendrá unas coordenadas (*u*, *v*, *w*) tales que $x=x_0+uU+vV+wW$. Estas coordenadas (*u*, *v*, *w*) se pueden hallar fácilmente usando álgebra lineal, como se muestra en la ecuación 36.

$$u = \frac{V \times W\ (X - X_0)}{V \times W \cdot U},$$
$$v = \frac{U \times W\ (X - X_0)}{U \times W \cdot V}, \qquad (36)$$
$$w = \frac{U \times V\ (X - X_0)}{U \times V \cdot W}$$

Cualquier punto que sea interior al sistema, verificará que para todas sus coordenadas (*u*, *v*, *w*) éstas se hallan en el intervalo [0, 1], de modo que basta con este cálculo para conocer qué vértices serán afectados por la deformación.

Al desplazar cualquiera de los puntos $P_{i,j,k}$ de su posición inicial en el entramado, pasaremos la coordenada dentro del espacio del mismo de cada uno de los puntos del objeto por la ecuación paramétrica 4 expuesta con anterioridad (recuérdese que, convenientemente, los valores de *u*, *v* y *w* se hallan en el intervalo [0-1]).

El vector resultante será la posición deformada del punto en las coordenadas cartesianas regulares.

Después de calcular la posición resultante para cada punto, se dibuja el modelo de la forma habitual. Se puede animar esta deformación simplemente cambiando las posiciones de los puntos de control del volumen de Bézier en cada frame.

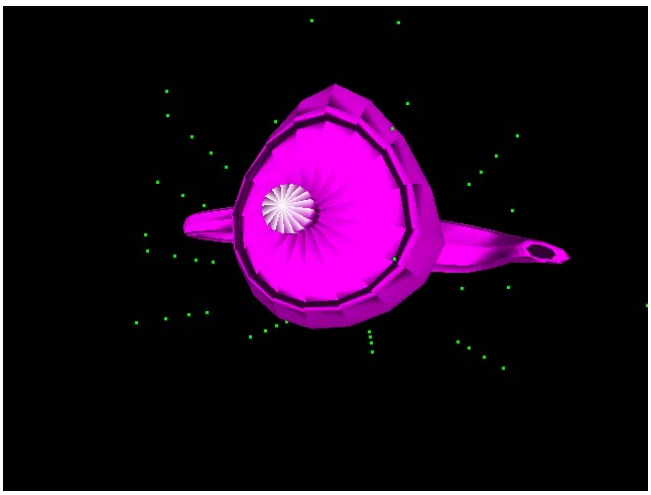


Figura 43: Modelo deformado con FFD global

En la fig. 43 podemos ver un objeto deformado desplazando manualmente los puntos de control.

Deformaciones Locales

La técnica FFD se concibe a menudo como un método de deformación global para contraer o expandir objetos. No obstante, si sólo se desean deformar ciertas partes localizadas, esta aproximación no es apropiada. Además, la utilización de un único FFD sobre el objeto puede provocar que éste parezca "hecho de gelatina" (fig. 43). Esto no es deseable para la mayoría de las situaciones, incluyendo la del caso de estudio.

Sin embargo, también puede concebirse la FFD como una técnica local para centrar esa deformación únicamente en ciertas partes del objeto.

Una forma de localizar dicha deformación sería aumentar el grado (y con ello los puntos de control del FFD) sin embargo, como ya se comenta en [13], los cambios siguen siendo globales y por tanto afectarán a todos los puntos del objeto, aunque según se aumente el número de puntos de control se localizará más el grueso de la deformación. Aun así, el coste computacional de esta deformación aumenta enormemente, ya que con un bloque FFD de grado n el bucle interno se itera n3 veces por vértice.

La solución pasa por utilizar más de un bloque FFD por objeto, ubicándolos convenientemente. En la figura 41 se aprecia un volumen FFD aplicado únicamente sobre una parte del modelo.

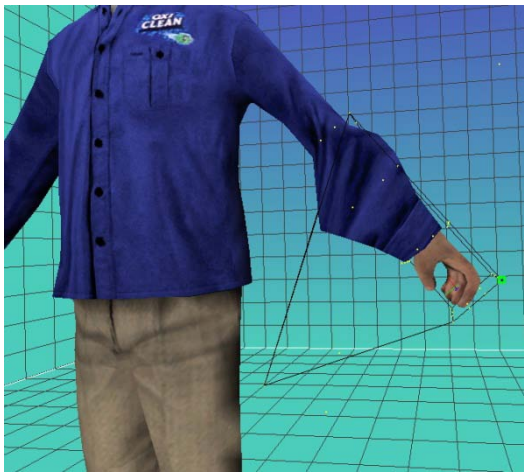


Figura 44: deformación localizada en el brazo

En el caso de estudio, la intención es dotar a un personaje, construido como un sistema óseo de deformación de la malla (y al que hemos dotado de capacidades de cinemática inversa usando la técnica Cyclic-Coordinate Descent [1],[7]) de elasticidad, por ejemplo, en sus extremidades: el personaje podrá estirar un brazo (p. ej. para alcanzar un objeto lejano) y al descargarse la tensión elástica éste se comportará de la forma esperada volviendo a su situación de reposo de una forma "intuitivamente consistente".

Para ello se define un volumen de FFD que envuelva, lo más ajustadamente posible los vértices de los miembros en cuestión, en la fig. 45 se representa el modelo de un personaje, cuya parte inferior del brazo ha sido envuelta en un volumen cúbico de Bézier, y en el que los puntos de control se han representado con puntos amarillos.

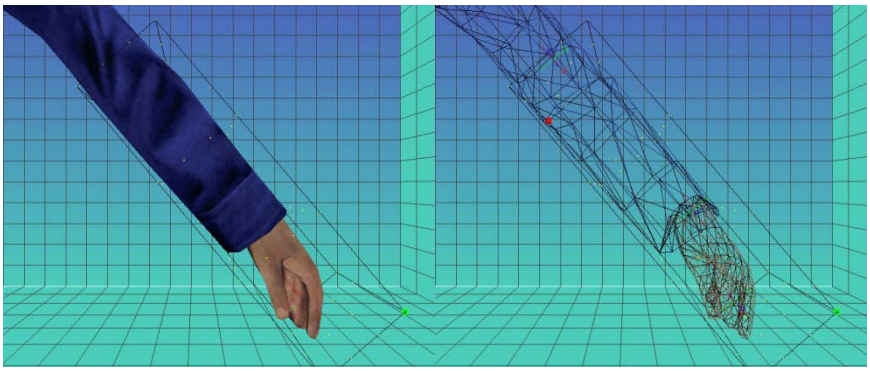


Figura 45: Brazo envuelto en volumen de deformación

Dado que la distancia al punto de control desplazado dentro del entramado del FFD (es decir, el valor [0, 1] de sus coordenadas *u*, *v, w*) es la base de la influencia de éste sobre cada vértice, es importante que el volumen envuelva lo más ajustadamente posible a los vértices de interés, pues cuanto mayor sea el entramado menos perceptible será la deformación.

8.4.2. Aplicación de elasticidad

Pero el caso de estudio planteado no trata de poder manejar los puntos de control de forma individual; al contrario, lo ideal es que los puntos de control sean transparentes al usuario. El elemento elástico (el brazo en este caso) debe estirarse o contraerse con un desplazamiento automático de los puntos de control que, implícitamente y a través del FFD, desplace los píxeles afectados. El usuario, por ejemplo, podría hacer clic [36] en un elemento del escenario (un objeto lejano al personaje) y que este automáticamente, y gracias también a la IK, se estirase hasta lo que su cuerpo le permitiera para luego aplicarse el comportamiento elástico del brazo y alcanzarlo.

Gracias a la cualidad de la técnica FFD de abstraer la superficie de control del objeto deformado, dotar de elasticidad al sistema pasa únicamente por aplicar la ecuación de elasticidad escogida sobre los puntos de control del entramado FFD: para el objeto, la aplicación de la elasticidad es totalmente transparente, no se trabaja directamente sobre los vértices del modelo si no sobre el entramado de deformación.

Oscilador armónico amortiguado

Como base para dotar de elasticidad el comportamiento de la deformación se ha escogido un oscilador armónico basado en la Ley de Hooke, por su simplicidad y sus resultados apropiados para esa idea de “consistencia intuitiva” de la que se hablaba con anterioridad.

Un oscilador armónico es un sistema que, al desplazarse sobre su posición de equilibrio, experimenta una fuerza restablecedora, *F*, proporcional al desplazamiento, x, tal y como enuncia la Ley de Hooke. En la introducción de esta memoria se profundiza más en la teoría de los osciladores armónicos.

En este caso es necesario que el brazo converja, una vez estirado y al cesar la fuerza tensora sobre el brazo, a su posición de reposo, en lugar de permanecer oscilando como lo haría un muelle regido por un movimiento armónico simple, de modo que es correcto decir que nuestro brazo será un oscilador armónico amortiguado, cuya amortiguación se definirá de forma constante (como un coeficiente, K_d).

Además, este factor de amortiguación permite garantizar la convergencia de la solución numérica.

Integración

Basándose en este modelo, calcularemos la aceleración de cada punto de control considerando éstos con masa unidad y gracias a la mencionada ley de Hooke, en función del desplazamiento sobre su posición original (ec. 2).

Calculada la aceleración de cada punto de control, se precisa ahora determinar para cada instante la posición en el espacio de dichos puntos de control del entramado para animar el comportamiento elástico del mismo. Dado un punto de control P_i en un instante de tiempo t se desea conocer la posición de P_i en el instante $t + \Delta$ suponiendo que se conocen qué fuerzas actúan sobre la partícula en ese periodo de tiempo.

La posición de cada punto de control en el espacio estará regida en última instancia por la segunda ley de Newton:

$$f = m * a \qquad (36)$$

Donde f es la fuerza, m es la masa y a es la aceleración, es decir, la segunda derivada de la posición, x.

Se precisa ahora determinar la posición de los puntos de control del entramado para animar el comportamiento elástico del mismo. Dado un punto de control P_i en un instante de tiempo t se desea conocer la posición de P_i en el instante $t + \Delta$ suponiendo que se conocen qué fuerzas actúan sobre la partícula en ese periodo de tiempo.

Para resolver la ecuación diferencial de segundo orden que determina la posición en el tiempo de cada punto de control, se puede representar dicha ecuación como dos ecuaciones diferenciales de primer orden y resolverlas con métodos estándar como la integración de Euler, Runge-Kutta [16][31] o Verlet [31][17], basada en la ecuación diferencial de segundo orden. La elección de uno u otro método de integración es un equilibrio entre el tiempo de computación y la precisión de integración. La alternativa escogida en este caso ha sido la integración de Verlet por sus buenos resultados en rendimiento y su sincronización en todo momento entre velocidad y desplazamiento.

Integración de Verlet

Cada punto de control de la retícula de deformación tundra asociadas dos variables: su posición x y su velocidad v. En cada iteración del ciclo de renderizado de la escena (es decir, en cada generación de fotograma), para calcular la nueva posición x’ y la nueva velocidad v’, se suelen efectuar los cálculos aplicando las reglas:

$$x' = x + v * \Delta t \qquad (37)$$

$$v' = v + a * \Delta t \qquad (38)$$

Donde *Δt* es el tiempo que se considera transcurre entre cada frame (paso temporal), y *a* es la aceleración, calculada como se ha visto utilizando la segunda ley de Newton (*f=m*a*). Esta es la integración simple de Euler.

En este caso, se ha preferido una representación que no hace uso de la velocidad de cada punto de control, realizando el paso de integración de forma diferente: en lugar de almacenar la posición y velocidad de cada punto de control, almacenamos su posición actual *x* y su posición previa, x_0, para calcular la nueva posición *x'*. Manteniendo fijo el paso temporal entre fotogramas, la regla de integración para cada fotograma es:

$$x' = 2 * x - x_0 + a * \Delta t^2$$
$$x_0 = x \qquad (39)$$

Esta técnica se conoce como integración de Verlet [45] y es un método muy estable, debido a que la velocidad se pasa implícitamente, por lo que es más difícil que la velocidad y la posición pierdan sincronización. En realidad, la fórmula $2x-x_0=x+(x-x^*)$ no es sino una aproximación de la velocidad actual (es la distancia recorrida en el último incremento temporal). No es la solución más precisa pero sí rápida y estable.

Para incluir dentro del modelo de integración la amortiguación del oscilador armónico (para representar la pérdida gradual de velocidad en el muelle debido a, por ejemplo, la resistencia del aire), una forma sencilla pasa por multiplicar el vector velocidad implícito ($x-x_0$) por una cantidad oscilante entre 0 y 1, K_d. Cuanto más próxima esté esta cantidad a 0 mayor deceleración (amortiguación) tendrá el sistema. Para invertir este valor simplemente lo restamos a 1 antes de efectuar la operación, de tal modo que el paso de integración quedará como el algoritmo siguiente.

```
pos = pos + (pos-old_pos)*(1.0-Kd) + aceleración*TIEMPO*TIEMPO
```

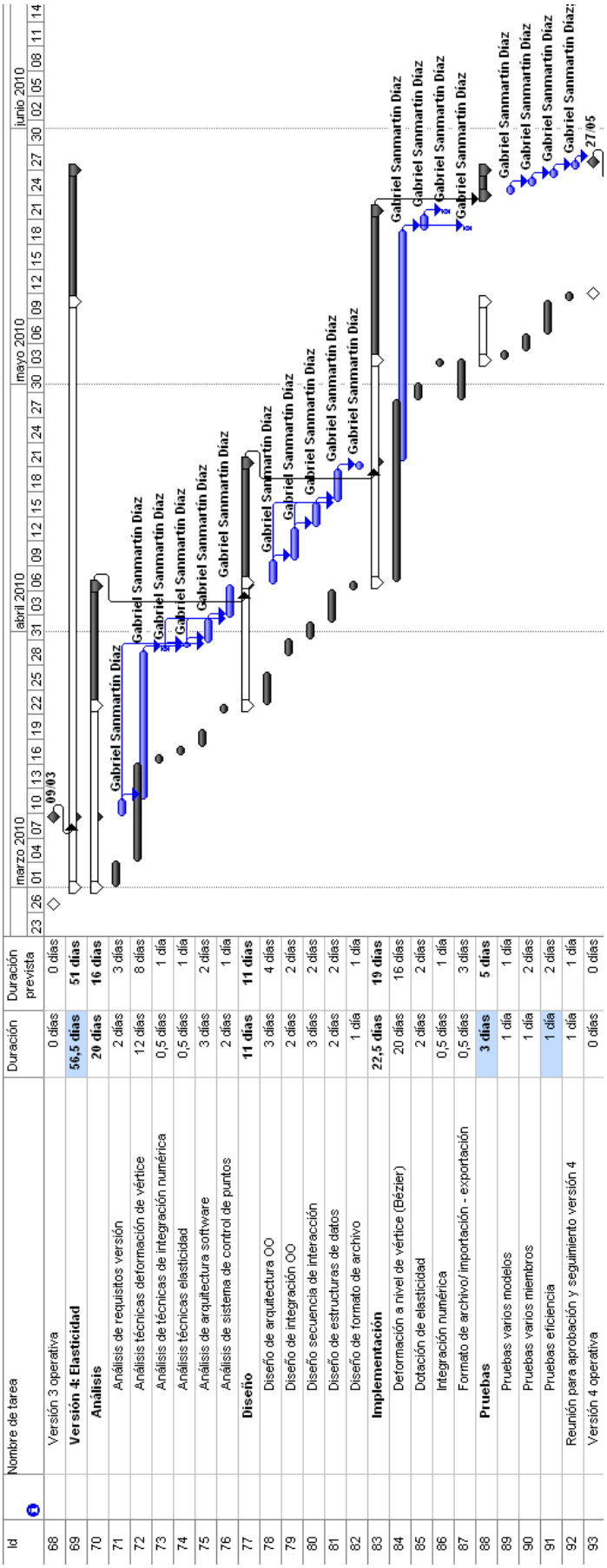
8.5. Seguimiento del proyecto

Tras el retraso en la entrega de la revisión anterior, la versión 4, lejos de amortiguar los efectos de aquélla, contribuyó al retraso global del proyecto provocando un retraso total de 13 días.

De forma similar a la revisión anterior, las causas del retraso fueron la necesidad de investigar las diferentes técnicas existentes para la deformación a nivel de pixel, y la dedicación excesiva a la técnica de muelles y masas, que finalmente hubo de ser desechada a favor de Free-Form Deformation.

Esta decisión comportó un retraso combinado en las fases de análisis e implementación de unos 7 días, compensados de nuevo gracias a una valoración demasiado permisiva de la fase de pruebas, que finalmente proporcionó cierta holgura en su ejecución.

La entrega de esta revisión se realizó el 27 de mayo de 2010.



Id	Nombre de tarea	Duración	Duración prevista
68	Versión 3 operativa	0 días	0 días
69	**Versión 4: Elasticidad**	**56,5 días**	**51 días**
70	**Análisis**	**20 días**	**16 días**
71	Análisis de requisitos versión	2 días	3 días
72	Análisis técnicas deformación de vértice	12 días	8 días
73	Análisis de técnicas de integración numérica	0,5 días	1 día
74	Análisis técnicas elasticidad	0,5 días	1 día
75	Análisis de arquitectura software	3 días	2 días
76	Análisis de sistema de control de puntos	2 días	1 día
77	**Diseño**	**11 días**	**11 días**
78	Diseño de arquitectura OO	3 días	4 días
79	Diseño de integración OO	2 días	2 días
80	Diseño secuencia de interacción	3 días	2 días
81	Diseño de estructuras de datos	2 días	2 días
82	Diseño de formato de archivo	1 día	1 día
83	**Implementación**	**22,5 días**	**19 días**
84	Deformación a nivel de vértice (Bézier)	20 días	16 días
85	Dotación de elasticidad	2 días	2 días
86	Integración numérica	0,5 días	1 día
87	Formato de archivo/ importación - exportación	0,5 días	3 días
88	**Pruebas**	**3 días**	**5 días**
89	Pruebas varios modelos	1 día	1 día
90	Pruebas varios miembros	1 día	2 días
91	Pruebas eficiencia	1 día	2 días
92	Reunión para aprobación y seguimiento versión 4	1 día	1 día
93	Versión 4 operativa	0 días	0 días

Figura 46: Seguimiento del proyecto a la entrega de la versión 4

9. Versión 5: Integración y Optimización

9.1. Objetivos

Si hasta el momento se han creado varias técnicas de deformación a nivel de vértice (deformación ósea + cinemática inversa y Free-Form Deformation) el objetivo de esta revisión es integrar ambas de tal manera que en lugar de ofrecer deformaciones independientes por separado se comporten de forma conjunta y dependientes la una de la otra; es decir, proporcionar una conjunción de la deformación ósea con la deformación elástica de tal modo que esta última pueda utilizarse, de forma específica, para dotar de elasticidad a ciertas partes de personajes (esto es, de ciertas extremidades o huesos).

Para ello será necesario establecer cierta asociación entre los huesos del personaje y las deformaciones elásticas que hayan podido crearse, entendiendo éstas como una revisión integrada de las creadas en la versión anterior. Asimismo, bajo esta nueva concepción, habrá de corregirse las posibles inconsistencias visuales surgidas de la combinación de ambas.

Por otro lado, se plantea la opción de optimizar el sistema de representación mediante el sistema de arrays entrelazados de vértices, normales y coordenadas de textura, para su trabajo directo sobre los procesadores vectoriales de las GPUs.

Representa pues esta revisión las correcciones finales del programa integral.

9.2. Análisis

9.2.1. Requisitos funcionales

Código requisito	Nombre requisito	Descripción de requisito
RV05F01	Comportamiento conjunto	Todo el sistema debe integrarse para la correcta manipulación y acceso desde el perfil del programador.
RV05F02	Cominación de esqueleto y FFD	El esqueleto provisto para la deformación ósea y los elementos deformables mediante FFD deberán comportarse de forma conjunta sin inconsistencias visuales derivadas por la acción de uno u otro, o por desincronización de las mismas.

Tabla 13: Requisitos funcionales de la versión 5

9.2.2. Requisitos no funcionales

Código requisito	Nombre requisito	Descripción de requisito
RV05NF01	Integración de las técnicas empleadas	Integración a nivel OO de las técnicas empleadas en una interfaz de control común.
RV05NF02	Optimización general del sistema	Optimización de los cálculos desempeñados por cualesquiera de

	las técnicas, desde la deformación de píxel hasta la representación gráfica.

Tabla 14: Requisitos no funcionales de la versión 5

9.3. Diseño

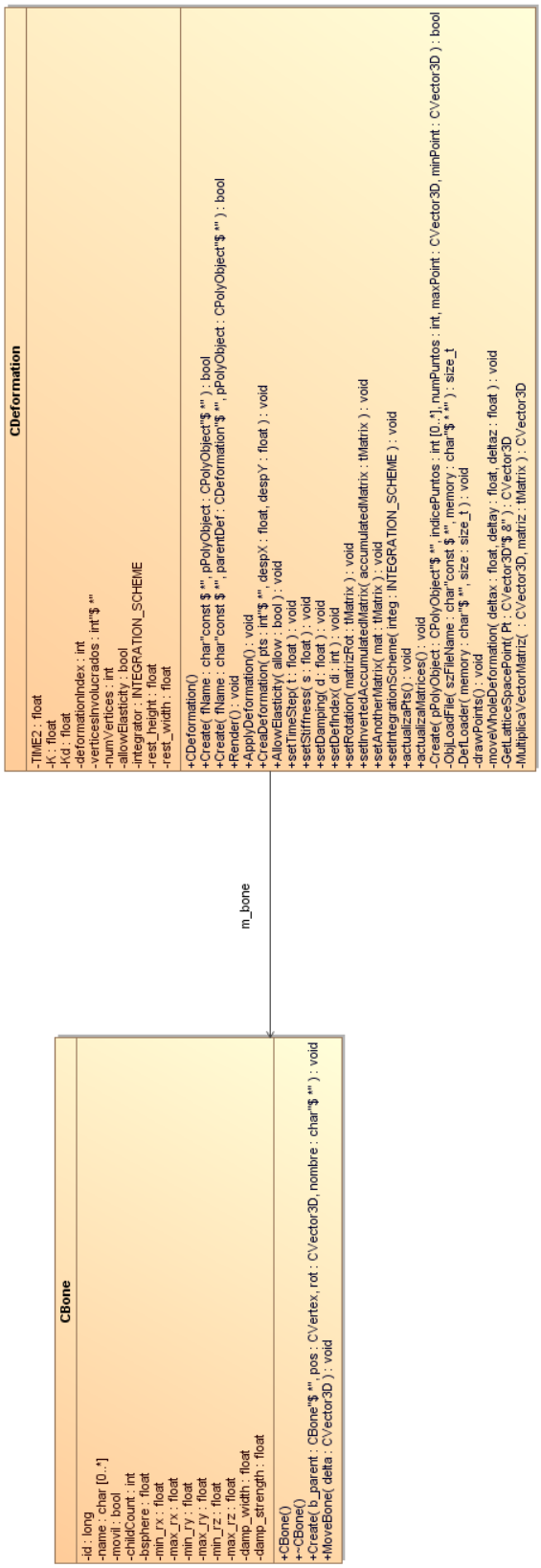


Figura 47: Diagrama de clases modificado para la versión 5

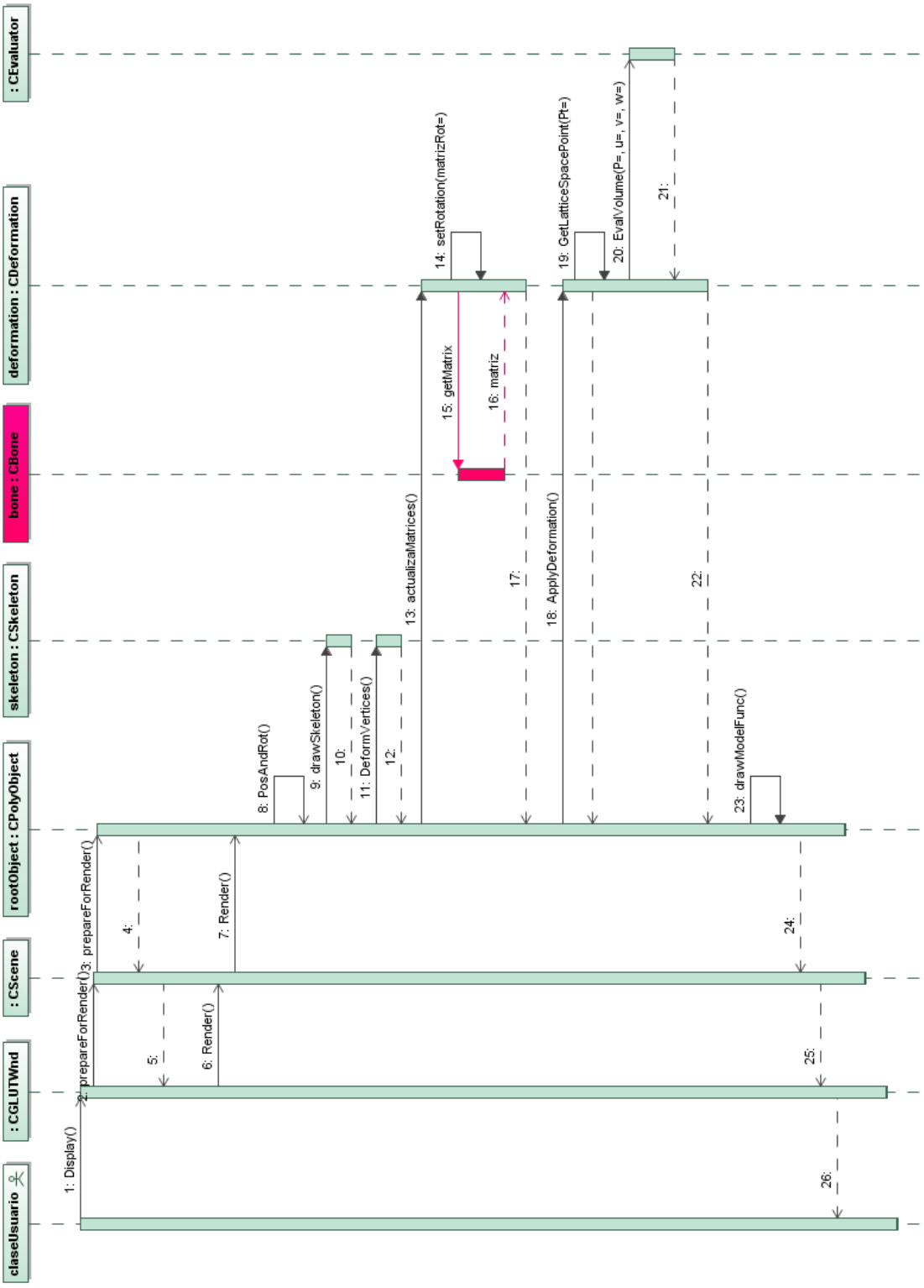


Figura 48: Modificaciones al diagrama de secuencia del caso de uso renderizar escena para la versión 5

9.4. Implementación

9.4.1. Combinación con sistema óseo

Una vez aplicado un volumen FFD sobre un área del modelo, por ejemplo, el brazo, si se ubica el entramado directamente sobre los vértices de interés, como se muestra en la figura anterior, puede ocurrir que el personaje, como consecuencia de una animación o de la aplicación de la cinemática inversa, desplazase dicha extremidad. En este caso las posiciones de los vértices del brazo se habrían desplazado fuera del entramado, con lo que perderíamos por completo toda deformación sobre dichos puntos, convirtiendo así el FFD en inútil (fig. 49b).

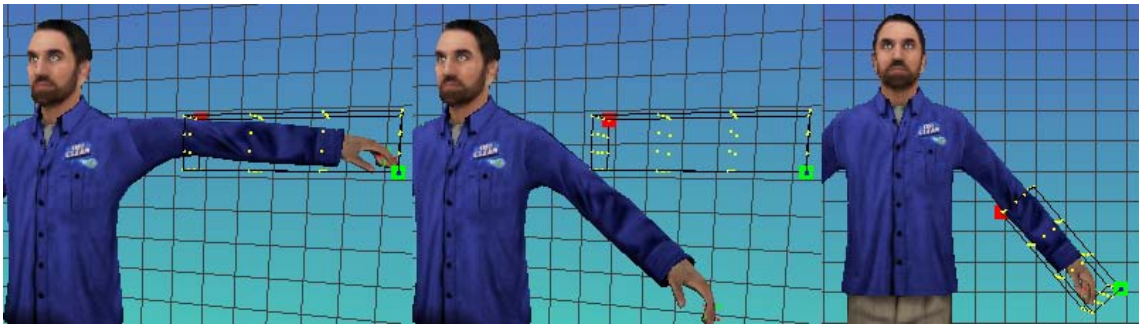


Figura 49: Modelo y FFD en reposo (izda.), FFD sin asociar al hueso (centro) y sincronizado con éste (dcha.)

Una posible solución a este problema sería recalcular la posición del entramado para moverla de acuerdo al movimiento del elemento de interés que va a ser deformado (en este caso, el hueso del brazo).

Ello pasa por, antes de aplicar la deformación, combinar la técnica de deformación ósea con la deformación elástica provista por el FFD: es necesario asociar este la retícula de deformación al hueso de la parte anatómica del personaje (hueso) sobre la que ejerce influencia, es decir, el brazo.

Como se ha visto en la versión 2 del sistema, cada hueso i tendrá un sistema de coordenadas (o matriz de transformaciones) asociado T_i^F y T_i^W, siendo T_i^F un sistema de coordenadas local definido de acuerdo al hueso inmediatamente anterior en la jerarquía, y T_i^W el sistema de coordenadas absoluto en la escena, verificándose la ecuación 40.

$$T_i^W = \prod_{j=0}^{i} T_j^F, T_1^F = T_1^W \qquad (40)$$

Por tanto, para ubicar el FFD en el hueso concreto bastaría con multiplicarlo por la matriz T_i^W del hueso al que se la asocia.

Aunque esto se ha hecho para, a nivel visual, poder mostrar siempre los puntos de control alrededor del brazo y moverlos acorde al mismo, siempre alineados con éste y en perfecta sincronización con sus movimientos (figs. 49c, 50), a la hora de calcular las deformaciones existe una alternativa más sencilla.

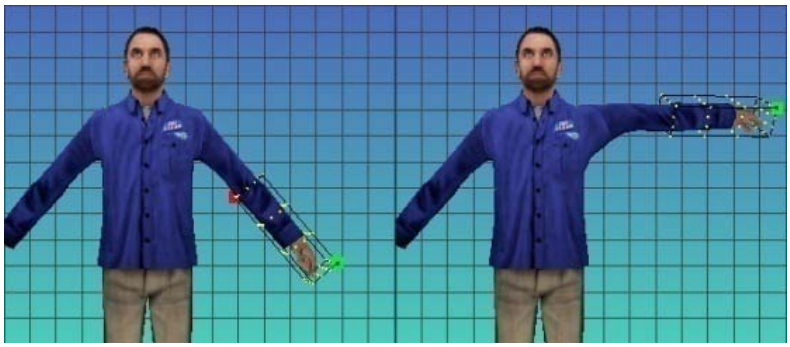


Figura 50: FFD en sincronización con el brazo

Ubicaremos siempre la retícula de deformación en el punto *(0, 0, 0)*, y siempre hacia direcciones (*x*, *y*, *z)* positivas, y obligar a la parte implicada (los vértices asociados al hueso en particular) a trasladarse a dicha posición antes de efectuar su transformación al sistema de coordenadas de FFD y el cálculo de su deformación. Esto puede verse en la figura 51.

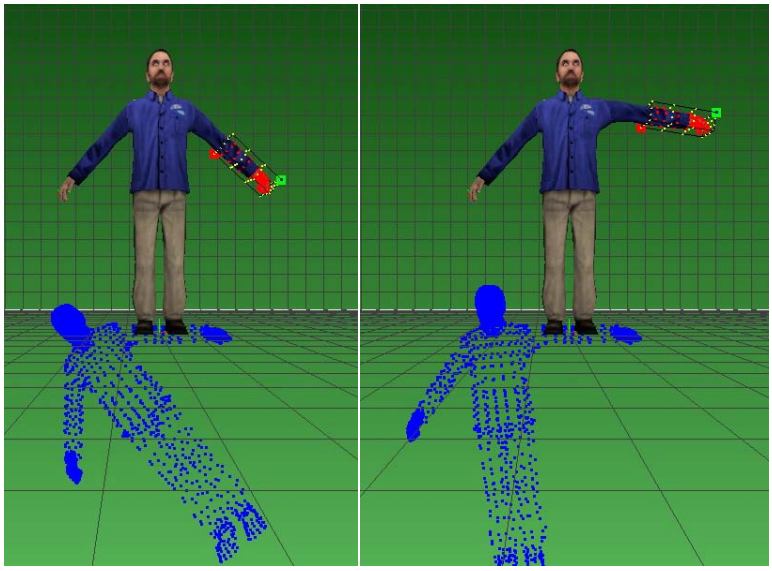


Figura 51: Para calcular su deformación, se fuerza al brazo a mantenerse en el origen

En este caso especial, cualquier punto del objeto puede transformarse de forma sencilla a las coordenadas [0, 1] del volumen (*u, v, w*) en el "punto del espacio de entramado". La ecuación 5 se convierte ahora para cada una de las coordenadas en la ecuación 41.

$$u = \frac{x}{x_{max}}, v = \frac{y}{y_{max}}, w = \frac{z}{z_{max}} \qquad (41)$$

Con ello que se simplifica el cálculo, de tal forma que la transformación al sistema de coordenadas del volumen de deformación es más sencilla, reduciéndose al siguiente pseudocódigo:

```
Vector3 PuntoEntramado(Vector3 Punto)
{
  Vector3 RelPos = Punto - Origen;
  Ret.x = RelPos.x / Tamano.x;
  Ret.y = RelPos.y / Tamano.y;
  Ret.z = RelPos.z / Tamano.z;
```

```
return Ret;}
```

Para trabajar, como decimos, siempre sobre el origen y en coordenadas positivas, se deberá multiplicar los puntos asociados al volumen FFD por la inversa de la matriz T_i^W, de tal modo que, siendo x_k el vértice *k*-ésimo involucrado en la deformación, e *i* el hueso al que se asocia ésta, su posición partiendo del origen x_{zero} puede calcularse siguiendo la ecuación 42.

$$x_{zero} = x_k * (T_i^W)^{-1} \qquad (42)$$

Supongamos ahora que por efecto de una solución IK particular, el brazo se dobla de la forma que muestra la figura 52.

Existe un número de vértices que, al haberse reubicado en el interior del entramado de la deformación (en la figura marcados en rojo), pasarían a ser afectados por ella, lo cual puede provocar que áreas del personaje que por la naturaleza de la deformación no deban deformarse sí salgan efectivamente deformadas tras aplicar la misma; no basta pues con estudiar para cada vértice si cae dentro del espacio del entramado (es decir, si sus coordenadas *u, v, w* tienen valores en el intervalo [0,1]), se debe especificar qué vértices son explícitamente los deformables lo cual además aumentará considerablemente la eficiencia al no tratarse todos los vértices que componen el modelo.

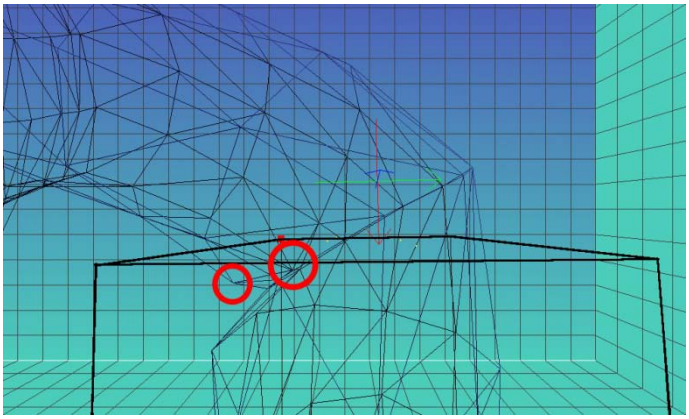


Figura 52: Vértices no deseados dentro de la región de interés

Esto puede hacerse de dos formas; la primera consiste en tomar la información del sistema óseo. Durante el proceso de creación de un sistema óseo se llama “skinning” a la asignación de porcentajes de influencia de un hueso sobre cada uno de los vértices que componen el objeto, de tal forma que si éste se desplaza de su posición original sólo se deformen los vértices que se conciben como afectados por el mismo. Dado que la retícula FFD ya ha sido asignada a un hueso dentro del sistema, se utiliza esta información para conocer los pesos de cada vértice y por tanto los vértices sobre los que el hueso ejerce influencia, siendo estos los afectados por el FFD.

La segunda alternativa consiste en trabajar de forma específica para cada modelo de trabajo y estudiar explícitamente qué vértices sobre el mismo se desea que se deformen y almacenarlos, por ejemplo, en un archivo externo que será leído en tiempo de ejecución.

Jerarquía de deformaciones

Si se crea una deformación local para el antebrazo del personaje, de tal forma que se pueda estirar y contraer esta parte del brazo individualmente sin afectar al resto del modelo, puede ocurrir lo que se muestra en la fig. 53. La deformación llega a estirar los vértices pertenecientes a la zona de influencia de la retícula, pero no ejerce influencia alguna sobre los elementos o huesos inferiores en la jerarquía (los huesos del brazo y la mano), que no se desplazan como resultaría esperable.

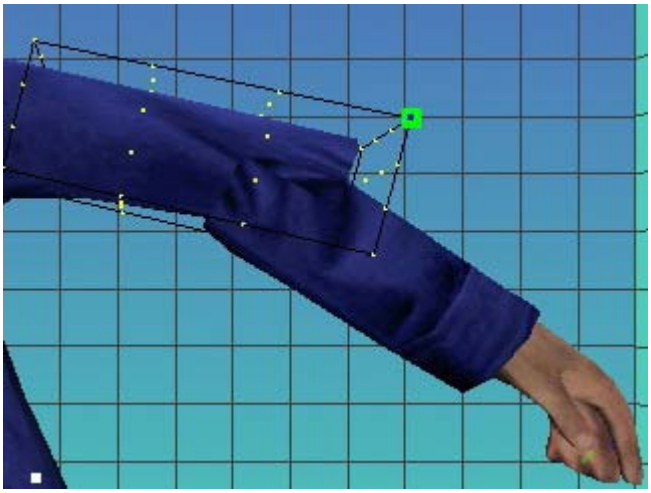


Figura 53: Deformación de vértices del hueso en una posición intermedia de la jerarquía

Para evitar esta inconsistencia visual, se crea un sistema jerárquico de deformaciones (fig. 54), similar al sistema óseo que se ha definido, de tal forma que mover el antebrazo implique no sólo mover estrictamente los vértices del mismo (a través de la influencia del FFD) sino de los objetos de la jerarquía que caigan a un nivel inferior, pero teniendo en cuenta que sólo se desea desplazar los vértices externos al entramado FFD, en lugar de deformarlos aplicando las ecuaciones estudiadas.

Es decir, no se deformarán utilizando los volúmenes de Bézier los vértices externos a la retícula (tampoco sería posible debido a que sus coordenadas en el sistema local definido por éstos no se sitúan en el intervalo [0,1] al caer fuera de la misma), sino aplicando una simple traslación, en la dirección apropiada (la del movimiento).

Esto resulta muy conveniente ya que en el caso de "elastizar" un miembro del personaje, la idea es restringir el movimiento a la dirección paralela al hueso, de tal modo que el desplazamiento simplemente debe tomar la dirección del vector definido por los puntos inicial y final del hueso en cuestión.

9.4.2. "VertexArrays"

Una optimización considerable en el sistema de representación de OpenGL consiste en la utilización de arrays de vértices, (punteros a los datos de representación como vértices, normales, coordenadas de textura, etc.) en lugar del sistema intuitivo y habitual de representar esta información en entidades o estructuras de datos separadas, como vectores, listas enlazadas, etc.

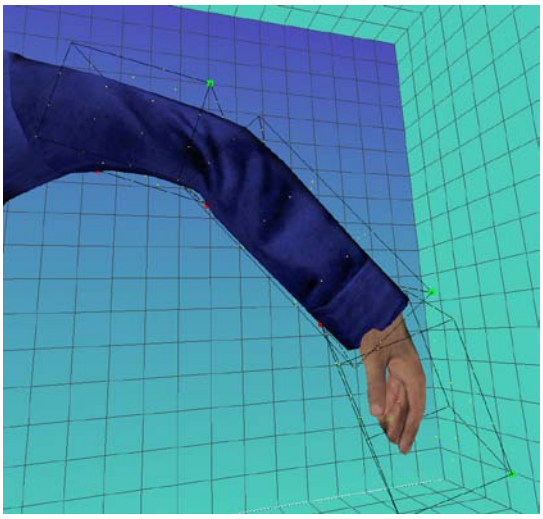


Figura 54: Jerarquía de deformaciones

Para ello, a la hora de cargar el modelo, en la estructura `vertexData`, un puntero a datos de tipo *float*, reservaremos memoria para los diferentes tipos de modelo con los que trabajaremos, en función de si contienen información de normales y coordenadas de textura o no, de la siguiente forma:

- El modelo contiene coordenadas de vértices (3), coordenadas de normales (3), y coordenadas de textura (2); el tamaño de cada ristra de datos es de 8 floats, y el formato de datos entendible por OpenGL es `GL_T2F_N3F_V3F`.

```
dataFormat = GL_T2F_N3F_V3F;
vSize = 8; //2 coord textura, 3 coord normales, 3 coord vertices
vertexData = (float *)malloc(sizeof(float) * vSize * fPos * vPerFace);
```

- El modelo contiene coordenadas de vértices (3) y coordenadas de normales (3), pero no coordenadas de textura; el tamaño de cada ristra de datos es de 6 floats, y el formato de datos entendible por OpenGL es `GL_N3F_V3F`.

```
dataFormat = GL_N3F_V3F;
vSize = 6; // 3 coord normales, 3 coord vertices
vertexData = (float *)malloc(sizeof(float) * vSize * fPos * vPerFace);
```

- El modelo contiene sólo coordenadas de vértices (3); el tamaño de cada ristra de datos es de 3 floats, y el formato de datos entendible por OpenGL es `GL_V3F`.

```
dataFormat = GL_V3F;
vSize = 3; // 3 coord vertices
vertexData = (float *)malloc(sizeof(float) * vSize * fPos * vPerFace);
```

Con esta información almacenada en la clase `CPolyObject`, es posible optimizar el dibujado de cada objeto en la escena, utilizando la primitiva `glInterleavedArrays()` de OpenGL, la cual recibe como parámetro el formato de datos que acaba de definirse, y el puntero a float

que se ha creado siguiendo dicho formato, y prepara en el pipeline de OpenGL los datos para ser representados posteriormente por `glDrawArrays();`
Esto se hace en la función `drawModelFunc()` de `CPolyObject`.

```
glInterleavedArrays(dataFormat,0,(GLvoid *)(data));

glDrawArrays(GL_TRIANGLES, pMesh->startIndex,pMesh->triangleCount * 3);
```

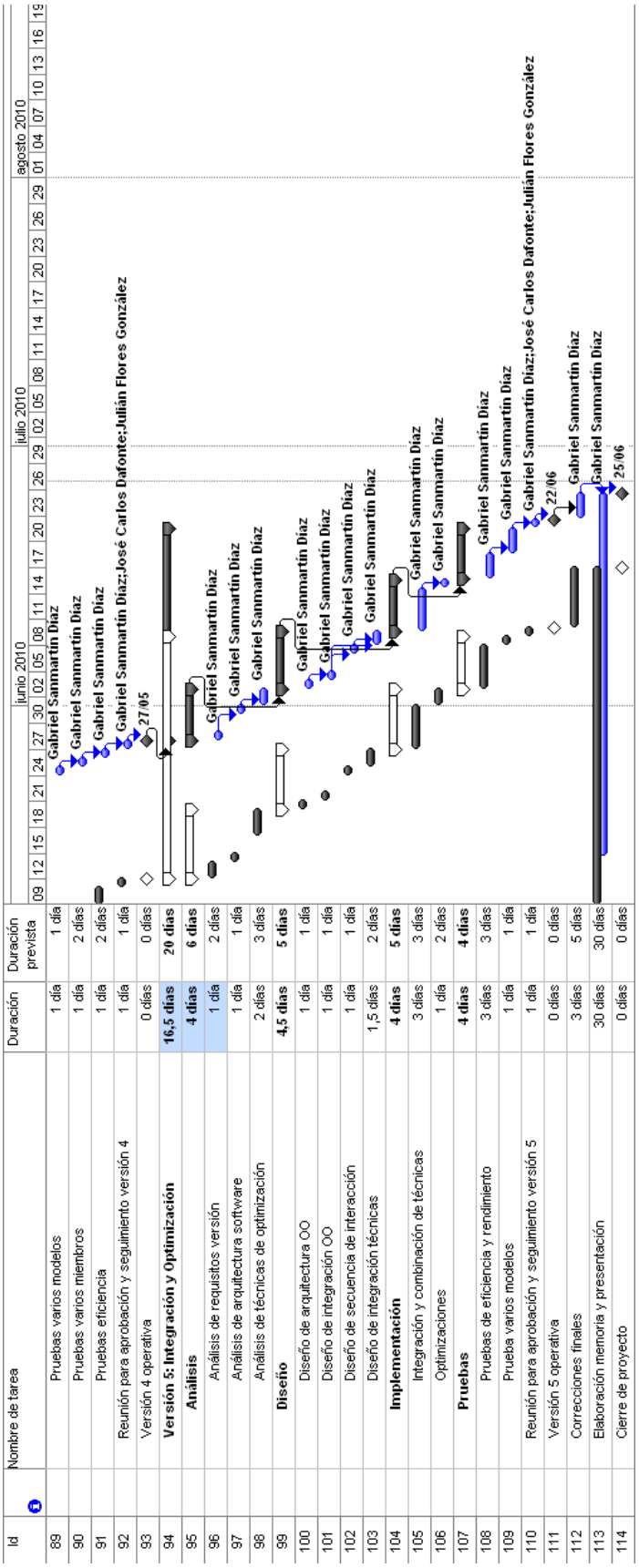
La razón de que esta técnica sea más eficiente que la convencional (recorrer una estructura de datos llamando a `glTexCoord()`, `glNormal3f()` y `glVertex3f()` en cada uno de los datos) se debe a que la información se presenta secuencialmente en el orden de aplicación dentro del backend de dibujado de la API, y en un formato común de datos, legible de manera directa (sin transformación) por parte de los procesadores vectoriales de la GPU.

9.5. Seguimiento del proyecto

Finalmente, el desarrollo de la última revisión partía con un retraso considerable y con la dificultad de una fecha límite cada vez más cercana, por lo que hubo de aumentar considerablemente el esfuerzo en este último incremento, que por fortuna comprendía una carga de trabajo mucho menor.

De este modo pudieron ahorrarse 4 días respecto a la duración planificada, gracias a que las planificaciones realizadas para las fases de análisis y en especial implementación eran bastante ajustadas, pero pudieron reducirse en su ejecución gracias a un pequeño aumento en la dedicación del alumno en la elaboración de las tareas (de 30 mins a 1 hora al día).

El retraso final en el momento de cierre de proyecto fue de 9 días, produciéndose este el día 25 de junio de 2010.



Id	Nombre de tarea	Duración	Duración prevista
89	Pruebas varios modelos	1 día	1 día
90	Pruebas varios miembros	1 día	2 días
91	Pruebas eficiencia	1 día	2 días
92	Reunión para aprobación y seguimiento versión 4	1 día	1 día
93	Versión 4 operativa	0 días	0 días
94	**Versión 5: Integración y Optimización**	**16,5 días**	**20 días**
95	**Análisis**	**4 días**	**6 días**
96	Análisis de requisitos versión	1 día	2 días
97	Análisis de arquitectura software	1 día	1 día
98	Análisis de técnicas de optimización	2 días	3 días
99	**Diseño**	**4,5 días**	**5 días**
100	Diseño de arquitectura OO	1 día	1 día
101	Diseño de integración OO	1 día	1 día
102	Diseño de secuencia de interacción	1 día	1 día
103	Diseño de integración técnicas	1,5 días	2 días
104	**Implementación**	**4 días**	**5 días**
105	Integración y combinación de técnicas	3 días	3 días
106	Optimizaciones	1 día	2 días
107	**Pruebas**	**4 días**	**4 días**
108	Pruebas de eficiencia y rendimiento	3 días	3 días
109	Prueba varios modelos	1 día	1 día
110	Reunión para aprobación y seguimiento versión 5	1 día	1 día
111	Versión 5 operativa	0 días	0 días
112	Correcciones finales	3 días	5 días
113	Elaboración memoria y presentación	30 días	30 días
114	Cierre de proyecto	0 días	0 días

Figura 55: Seguimiento del proyecto a la entrega de la versión final

10. Conclusiones y trabajo futuro

Aunque es muy habitual en aplicaciones 3D usar animaciones previamente creadas para dotar de movimiento a los personajes, a menudo es necesaria una flexibilidad y ductilidad en las mismas que obliga a utilizar métodos algorítmicos para generarlas en tiempo real.

Se ha desarrollado una metodología de trabajo basado en un estudio completo de estas técnicas para la generación de animaciones para movimientos de tipo realista, sobre un humanoide. Para ello, a partir del bajo nivel proporcionado por la API de representación OpenGL, se ha creado un sistema integral de deformaciones y representación de objetos, comenzando por un gestor de escenas y objetos dentro de las mismas, con una representación jerárquica y óptima de los modelos 3D, y continuando con un esfuerzo centrado en la representación de personajes, ideal para el trabajo en videojuegos y aplicaciones lúdicas de animación.

El siguiente paso ha sido, de forma lógica, la representación de personajes 3D creando la abstracción de la deformación ósea, la cual estructura bajo la malla del persona un conjunto jerárquico de huesos para la deformación específica de cada extremidad y elemento articulable de un personaje, pudiendo de forma sencilla gestionarse animaciones y movimientos del mismo en un entorno realista.

La dotación de interactividad ha pasado por la implementación de métodos de cinemática inversa, estudiados intensivamente hasta hallar el método óptimo para los objetivos deseados, creándose por tanto un sistema interactivo en el que se generan animaciones on-the-fly con la intervención del usuario.

Sobre dicho humanoide se han generado una serie de volúmenes de Bézier que serán la base de deformadores FFD sobre el objeto. Para evitar inconsistencias visuales, se ha creado un sistema jerárquico de deformaciones, similar al sistema óseo que se ha definido, de tal forma que mover un elemento del mismo se vean afectados todos los asociados al mismo. Como patrón de la deformación y para dotar de esa sensación de elasticidad marcada como objetivo primario, se ha utilizado un oscilador armónico simple. Esta estructura da lugar a una deformación intuitivamente coherente del humanoide, en un proceso de deformación se realiza de forma transparente al usuario y en tiempo real.

Este estudio intensivo de múltiples técnicas de representación y deformación ha servido al alumno para empaparse de las diferentes metodologías y paradigmas existentes para la gestión del comportamiento 3D, lo que le ha servido de enorme ayuda para el desempeño de su actual trabajo en el ámbito de estudio del proyecto.

El caso de estudio se ha probado en un PC Intel Core 2 Quad Q9450 con cuatro núcleos a 2.66 Ghz, 4Gb de RAM y una gráfica nVidia GeForce 8800GT de 512Mb dedicados, obteniéndose así un framerate medio superior a los 250 fps, para un modelo 3D de 2552 vértices y 4895 caras, utilizándose 3 deformaciones de ~1500 vértices cada una.

Si bien los resultados son estética y visualmente correctos, un posible trabajo futuro sería la aceleración y cálculo de las deformaciones de Bézier a través de la tarjeta gráfica empleando las matrices de deformación, de manera similar a como se hace con las deformaciones óseas del esqueleto del modelo. Esto cargaría el grueso de los cálculos FFD sobre la gráfica, aprovechando la conveniencia de que tanto para el sistema óseo como para los P_x del volumen FFD los valores serían oscilantes en el intervalo [0,1], tal y como se comenta en [4][6]. Es también una línea futura de trabajo la posible utilización de la GPU para los cálculos.

11. Bibliografía

[1] Ang, Jason. "*Direct Manipulation of Free-Form Deformations*", Winter 2000, (http://www.student.cs.uwaterloo.ca/~cs779/Gallery/Win2000/jang/index.html). Último acceso, mayo 2010.

[2] Apron tutorials. "*The Character Animation FAQ*" (http://www.morrowland.com/apron/article/general/character_animation/index.php) Último acceso, enero 2010.

[3] Badler, Norman et al. *"Simulating Humans: Computer Graphics Animation and Control",* Oxford University Press, 1993.

[4] Baraff, David, Witkin, Andrew. "*Large Steps in Cloth Simulation"*, ACM SIGGRAPH 98, Computer Graphics, p 43-51. July 1998.

[5] Chahal, Pardeep. "*Physically Based Model of Cloth Draping*", 2001 (http://members.shaw.ca/chahal/clothpaper.htm). Último acceso, febrero 2010.

[6] Chua, C., Ulrich, N. "*Hardware-Accelerated Free-Form Deformation*", ACM SIGGRAPH 2000, Computer Graphics, p 33-39.

[7] Coquillart, Sabine. "*Extended free-form deformation: a sculpturing tool for 3D geometric modeling*", ACM SIGGRAPH, Computer Graphics, v.24 n.4, p 187-196, Aug.1990

[8] DHPoware. *"OpenGL OBJ Viewer"* (http://www.dhpoware.com/demos/glObjViewer.html). Último acceso, diciembre 2009.

[9] Elias, Hugo. "*Improved methods for Inverse Kinematics"*, 2004, (http://freespace.virgin.net/hugo.elias/models/m_ik2.htm). Último acceso, marzo 2010.

[10]Featherstone, R. "*Robot Dynamics Algorithms*". 1987, Boston: Kluwer. ISBN 0-89838-230-0.

[11]Fêdor, Martin. "*Application of inverse kinematics for skeleton manipulation in real-time*", Proceedings of the 19th spring conference on Computer graphics, April 24-26, 2003, Budmerice, Slovakia.

[12]Feng, J., Nishita, T., Jin, X. and Peng, Q. "*B-spline free-form deformation of polygonal object as trimmed Bézier surfaces"*, November 6, 2002, The Visual Computer, pg 493-510.

[13] Ferrier, Alex. "*Real-Time Soft-Object Animation using Free-Form Deformation*" (http://www.gamasutra.com/view/feature/3372/realtime_softobject_animation_.php). Último acceso, mayo 2010.

[14]Hecker, Chris. *"My adventure in Inverse Kinematics"*, 2002 Game Developers Conference (http://chrishecker.com/images/7/76/Gdc2002-ik.ppt)

[15]J3d.org. "*The Matrix and Quaternions FAQ*", 30th November 2003, (http://www.j3d.org/matrix_faq/matrfaq_latest.html). Último acceso, junio 2010.

[16]Jack, Hugh. "*Runge-Kutta Integration*", 2001 (http://claymore.engineer.gvsu.edu/eod/refer/refer-52.html). Último acceso, diciembre 2009

[17]Jakobsen, Thomas. "*Advanced Character Physics*". Proceedings of Game Developers Conference, 2001. p 1-10.

[18]Lander, Jeff. *"Hierarchy of different objects",* Gamasutra article, 1997 (http://darwin3d.com/gamasutr.htm). Último acceso, junio 2010.

[19]Lander, Jeff. "*Cloth Simulation using Mass and Spring System"*, May 1999, Game Developers Conference (http://www.darwin3d.com/gamedev/articles/col0599.pdf). Último acceso, junio 2010.

[20]Lander, Jeff. "*Hardware Accelerated Free-Form Deformation*", June 2000, Game Developer Magazine. Último acceso, junio 2010.

[21]Lander, Jeff. "Making kine more flexible", November 2008, Game Developer Magazine (http://www.darwin3d.com/gamedev/articles/col1198.pdf). Último acceso, junio 2010.

[22]Lander, Jeff. "*Notes on 3D IK*", 1998, Game Developer Magazine (http://www.darwin3d.com/gdm1998.htm#3DIKSTUFF). Último acceso, junio 2010.

[23]Lander, Jeff. *"Nothing but Skin and Bones: Creating compelling 3D characters"*, March 1998, IGDN, (http://www.darwin3d.com/conf/igdn0398/index.htm). Último acceso, junio 2010.

[24]Lander, Jeff. "*Oh my god, I inverted Kine!*", September 1998, Game Developer Magazine (http://darwin3d.com/gamedev/articles/col0998.pdf). Último acceso, junio 2010.

[25]Lander, Jeff. "*Over my dead, poligonal body",* October 1999, Game Developer Magazine (http://darwin3d.com/gamedev/articles/col1099.pdf). Último acceso, junio 2010.

[26]Lander, Jeff. "*Skin Them Bones",* May 1998, Game Developer Magazine (http://darwin3d.com/gamedev/articles/col0598.pdf). Último acceso, junio 2010.

[27]Lasseter, John. "*Principles of Traditional Animation Applied to 3D Computer Animation*" : July 1987. Proceedings of Siggraph '87. In Computer Graphics (Vol. 21, No. 4)

[28]Mandel, Michael. *"Moving Beyond Ragdolls: Generating Versatile Human Behaviors by Combining Motion Capture and Controlled Physical Simulation"*, 2005, Game Developers Conference

[29]Martin, Steve. "*Stable and Responsive Cloth*", Fall 2003, University of Berkeley (http://stevezero.com/eecs/cs294proj3/). Último acceso, mayo 2010.

[30]Matyka, Maciej. "*How to Implement a Pressure Soft Body Model"* March 2004, Carmel High School Computer Science (http://zigonstuff.com/chs0506/HowToSoftBody.pdf) Último acceso, mayo 2010.

[31]Mosegaard, J.: "*Realtime cardiac surgery simulation*", PhD Dissertation, Aarhus University Hospital, Denmark, 2004.

[32]Noe, Karsten. "*Free-Form Deformation using RBFs",* August 2007, (http://cg.alexandra.dk/tag/free-form-deformation/) Último acceso, junio 2010.

[33]Okino.com. *"Skeletons and Mesh Skinning Conversion - An Overview",* 2001 (http://www.okino.com/conv/skinning.htm). Último acceso, enero 2010.

[34]Terzopoulos, Demetri et al. "*Elastically Deformable Models*", Computer Graphics, Vol. 21, No. 4 (SIGGRAPH 1987): pp. 205-14

[35]The Game Programming Wiki. "*OpenGL:Tutorials:Basic Bones System"*, June 2006 (http://gpwiki.org/index.php/OpenGL:Tutorials:Basic_Bones_System). Último acceso, enero 2010.

[36]The Game Programming Wiki. "Picking with OpenGL", 2009, (http://gpwiki.org/index.php/OpenGL:Tutorials:Picking). Último acceso, octubre 2009.

[37]Thomas W. Sederberg, Scott R. Parry. "*Free-Form Deformation of Solid Geometrics Models*" ACM SIGGRAPH Computer Graphics, v.20 n.4, p 151-160, Aug. 1986

[38]Usal, Ilyas. "*Springs: Elasticity, Physics, and Flash MX"*, (http://www.kirupa.com/developer/actionscript/spring.htm). Último acceso, febrero 2009.

[39]Wang, L., C., Chen, C. "*A Combined Optimization Method for Solving the Inverse Kinematics Problem of Mechanical Manipulators*" IEEE Trans. On Robotics and Applications, vol. 7, no. 4, pp 489-499, 1991

[40]Farin, Gerald. "*Curves and surfaces for computer-aided geometric design (4 ed.)"*, 1997 Elsevier Science & Technology Books

[41]J.D. Foley et al. "*Computer Graphics: Principles and Practice in C (2nd ed.)",* 1992, Addison Wesley

[42]Marion, Jerry B. "*Dinámica clásica de las partículas y sistemas*" 1996, Ed. Reverté

[43]Pressman, Roger. "Ingenieria de Software Un Enfoque Práctico" (5th Ed), McGraw-Hill.

[44]PMBOK. *"Guía de los Fundamentos de la Dirección de Proxectos (Tercera Edición) – Guía del PMBOK"*. Norma Nacional Americana (ANSI/PMI 99-001-2004)

[45] Verlet, L. "*Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules*", 1967, Phys. Rev., 159, 98-103