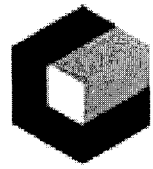

INSTITUTO SUPERIOR POLITÉCNICO
"JOSÉ ANTONIO ECHEVERRÍA"

FACULTAD INGENIERÍA INDUSTRIAL

ESPECIALIDAD INGENIERÍA INFORMÁTICA

Biblioteca Gráfica Para Sistemas de Realidad Virtual

Trabajo para optar por el Título de Ingeniería en Informática

Autores: Yanoski Rogelio Camacho Román

Fernando Jiménez López

Tutores: Msc. José Ignacio Guzmán Montoto

Ing. Lester Barro Gallardo

Ciudad de la Habana

2004

Resumen

Los Sistemas de Realidad Virtual (SRV), se han expandido considerablemente a diferentes sectores de la sociedad en los últimos años. Esta tecnología, ofreciendo prestaciones de una manera más intuitiva, segura y económica, puede encontrarse en aplicaciones de la defensa, la medicina, la educación y el entretenimiento.

Toda la tecnología de Realidad Virtual (RV), se apoya en un núcleo central, que se encarga de accionar los diferentes módulos para el funcionamiento del sistema. Este núcleo, también denominado "motor de simulación", cuenta entre sus módulos básicos con el sistema de gráfica tridimensional en tiempo real.

Este proyecto aborda el diseño de una biblioteca de clases para la gráfica 3D en C++, que constituye la base de un motor de RV para sistemas de simulación. Para alcanzar esta meta, se estudian otras bibliotecas existentes, y se trabaja en la investigación y desarrollo de algoritmos eficientes para la representación tridimensional en tiempo real. Se exponen además, los métodos utilizados para lograr la inserción y animación de personajes en los entornos virtuales. La herramienta de *software* resultante de esta investigación, permitirá mayor eficiencia al representar escenarios virtuales sobre computadoras personales y la reducción de los tiempos de desarrollo de Sistemas de Realidad Virtual.

Contenido

Introducción

Con el surgimiento y desarrollo de las técnicas de Realidad Virtual, la Informática Gráfica ha dado un salto en la calidad de sus aplicaciones, en cuanto a interactividad y realismo.

Los beneficios de la RV se dejan ver en la industria del entretenimiento (vídeo-juegos, efectos especiales en el cine, etc.); en la educación (sistemas multimedia); en la visualización científica de datos y de fenómenos no perceptibles por el ojo humano; en la medicina (realización de prótesis, intervención médica a niveles celulares, ingeniería genética, teleoperaciones, virtuoterapia,...); en la meteorología (estudio de tormentas eléctricas, impactos geológicos de volcanes en erupción...); en el diseño de compuestos químicos, análisis molecular, entre otras.

Al mismo tiempo, estas tecnologías han dado paso al surgimiento de los simuladores de Realidad Virtual, sistemas orientados fundamentalmente al desarrollo de habilidades en la manipulación de equipos, mediante una ambientación digital que logra un gran acercamiento a la realidad, con el ahorro de recursos de estudio y entrenamiento.

El ejército estadounidense es uno de los principales clientes de los Sistemas de Realidad Virtual, dándole un uso especial en los simuladores para el entrenamiento de los soldados que formarán parte de las fuerzas de élite, desarrollando la capacidad de tomar decisiones adecuadas en situaciones críticas. [17]

En Cuba, una de las empresas que ha impulsado el desarrollo de los SRV, es el "Centro de Investigación y Desarrollo #2" (CID2), conocido en el mercado como **SIMPRO** (Simuladores Profesionales). Esta empresa comenzó como proyecto desde el año 1994, y oficialmente existe con el nombre de SIMPRO desde el

año 2000. Cuenta con numerosos premios a las investigaciones y calidad de sus productos desde su fundación, entre ellos:

- Premio de la Academia de Ciencias a las investigaciones más destacadas del año en dos ocasiones. (1996, 1998)
- Dos premios relevantes en Forum Nacional de Ciencia y Técnica.
- Premios en la Feria Internacional del Transporte a la calidad del producto.

Para el desarrollo de sus simuladores, SIMPRO inicialmente empleaba motores de visualización comprados en el extranjero, teniendo que pagar licencias y versiones actualizadas aún cuando no respondieran del todo a sus necesidades, creando una fuerte dependencia tecnológica. Además, el acceso a herramientas de desarrollo y otros recursos informáticos es limitado, debido al bloqueo económico y comercial impuesto por Estados Unidos.

Esto les crea la **necesidad** de concebir herramientas de trabajo propias que permitan desarrollar productos con rapidez y calidad. Por otra parte, el tener un código propio en un tema como la simulación para la defensa, ofrece seguridad, aprovechamiento de los recursos humanos del país (presentes en las universidades), y garantiza la actualización con los últimos adelantos de la ciencia y la técnica.

El **problema** principal a resolver por este proyecto, es la ausencia de un motor para la visualización 3D en tiempo real de los entornos virtuales de la empresa. Otro problema específico a resolver es la necesidad de obtener mayor realismo incorporándole personajes animados.

A continuación se hará un primer acercamiento a las definiciones del objeto de estudio y el campo de acción en que se enmarca el problema, como puntos de partida para concebir la solución del mismo.

La empresa SIMPRO cuenta con un grupo de departamentos encargados de la investigación, diseño y producción de los componentes de los simuladores, tanto de software como electro-mecánicos.

Los procesos desarrollados en cada área y la interacción entre ellas, son el **objeto de estudio** de este proyecto, y el **campo de acción**, la parte relacionada con la visualización eficiente de SRV en tiempo real.

Este proyecto propone como **objetivo** general, obtener una biblioteca gráfica de tiempo real para SRV que soporte la deformación de mallas, las cualidades físicas y emocionales de personajes, modificadores de la escena (cualquier alteración de los datos de los objetos que provoque un cambio visual), la utilización transparente de varias librerías gráficas, y la flexibilidad a actualizaciones futuras.

Como objetivos específicos se pretende: el empleo de técnicas para la visualización de la escena, y la incorporación de animaciones con el realismo adecuado para estos sistemas.

Se plantean entonces un grupo de tareas que permitirán satisfacer los objetivos, y que se pueden resumir en las siguientes:

- Investigación de las características básicas de los motores de Sistemas de Realidad Virtual.
- Análisis de las potencialidades de las librerías gráficas OpenGL y DirectX.
- Investigación de algoritmos, tecnologías y tendencias actuales utilizados en la creación y visualización de los mundos virtuales.
- Desarrollo de soluciones técnicas para alcanzar los objetivos propuestos.
- Análisis y diseño de una biblioteca de clases que dé solución a los problemas planteados.

Como resultado de este trabajo, se pretende obtener un motor de RV actualizable, con soporte para diferentes situaciones que aparecen en el mundo real. Esta herramienta informática, permitirá al programador el rápido desarrollo de aplicaciones de RV, de manera que la preocupación no esté en **cómo** crear soluciones básicas, sino en **qué** hacer para resolver sus problemas específicos utilizando las soluciones aportadas por el motor.

El contenido de este trabajo se encuentra estructurado de la siguiente manera:

En un primer capítulo, "Fundamentación Teórica", se hace un análisis bibliográfico donde se investigan las características de los SRV, las técnicas, algoritmos y tendencias actuales para el desarrollo de motores de RV. En el capítulo 2, "Soluciones Técnicas", se exponen las características técnicas que presentará el sistema como solución a los problemas planteados. En el capítulo 3, "Descripción de la solución propuesta", se analiza con mayor profundidad el objeto de estudio, y se crea el modelo del dominio, se hace la captura de requisitos y se crean los modelos de casos de uso del sistema. En el capítulo 4, "Análisis y diseño del sistema", se presentan los diagramas de clases de análisis, de clases de diseño y de secuencia. En el capítulo 5, "Implementación del sistema" se muestran los diagramas de despliegue y de componentes. Finalmente, se ofrece un glosario de abreviaturas y uno de términos para ayudar a la comprensión del lenguaje técnico utilizado a lo largo del trabajo.

1

Capítulo 1 Fundamentación Teórica

Introducción

Los Sistemas de Realidad Virtual, exigen ante todo una rápida interacción con el usuario, por lo que se debe ser cuidadoso a la hora de seleccionar los algoritmos y métodos sobre los que se basarán dichos sistemas.

Existen determinadas características implícitas que debe cumplir todo Sistema de Realidad Virtual. Son, por citar algunos ejemplos, el soporte a distintos tipos de modelos, el tratamiento de colisiones, los efectos de luces y sombras, la posibilidad de aplicar texturas a los cuerpos, las vistas en distintas proyecciones, el control del tiempo, la reproducción de leyes físicas, los efectos ambientales, la reproducción de sonidos, e incluso, el soporte inteligente y la inserción de personajes animados.

En el presente capítulo se hace un análisis de las características básicas de los SRV, así como de los principales algoritmos, tecnologías y tendencias que se están empleando en el mundo para el trabajo con los componentes mencionados. Se analizan además las diferencias y funcionalidades de algunos motores de SRV, y los algoritmos y estructuras de clases que soportan sus funcionamientos.

1.1 Sistemas de Realidad Virtual

Existen varios autores que han intentado dar una definición a partir de la caracterización de lo que consideran ***Sistemas de Realidad Virtual***, por sólo citar algunos ejemplos se tiene:

–"Un SRV es aquel que da al usuario la experiencia de estar inmerso en un entorno sintético" (H.Fuchs). [8.5]

–"La RV se caracteriza por ofrecer la ilusión de participación en un entorno sintético en vez de una observación externa del mismo. La RV es una experiencia inmersita y multisensorial" (M. A. Gigante). [8.5]

–"Un entorno virtual se caracteriza por ser interactivo, con un procesamiento especial de imagen, sonido y tacto, para convencer al usuario de que se encuentra inmerso en un espacio sintético" (S.R.Ellis). [8.5]

A partir del estudio de las caracterizaciones anteriores, se puede concluir que un SRV podría definirse como un sistema informático que simula objetos, eventos o procesos del mundo real, basado en acciones en tiempo real, de la gráfica tridimensional, la acústica y el tacto. Esta tecnología permite la interacción y propone la inmersión de los sentidos en el mundo simulado.

Instantes importantes de la evolución histórica

Se señalarán algunos momentos que han marcado el desarrollo de los Sistemas de Realidad Virtual a nivel mundial. [8.5]

–Morton Heilig, "Sensorama", 1962: Recorrido simulado en motocicleta por N.Y. Tuvo todos los elementos de la RV excepto la interactividad: viento, vibración, película estéreo, sonido estéreo y olores.

–Ivan Sutherland, "Theultimatedisplay", 1965: Escribe sobre la interactividad, los dispositivos de realimentación y efectores sensoriales.

–Ivan Sutherland, "AHead Mounted 3D Display", 1968: Describe un dispositivo de representación estéreo con dos pantallas que superponían imágenes del mundo real con imágenes por computador.

–VIVED (*Virtual Visual Environment Workstation*, NASA), 1984. Sistema para astronautas.

–VIEW (*Virtual Interactive Environment Workstation*, NASA), 1986. Sistema completo.

A partir de este momento se vio una desaceleración en el desarrollo de los SRV, debido a que el desarrollo informático no pudo ser acompañado por el resto de los componentes físicos de los simuladores. [8.5]

Características de los SRV

Todo sistema de realidad virtual debe tener capacidad de generar las imágenes (tridimensionales) en tiempo real según la posición del usuario y las señales del exterior. [8.5]

Debe permitirse también la interactividad, o sea, que el usuario participe sobre el SRV a través de dispositivos de control, y pueda cambiar el estado de los objetos (agarrar, desplazar, etc.) y navegar por el entorno (cambiar el punto de vista, hacer colisiones virtuales). [8.5]

La ilusión de realidad es una característica muy importante de los SRV, y se basa en la estimulación física (estimulación corporal, estereoscopía, localización sónica, etc.) y en la estimulación psicológica (credibilidad del mundo virtual, interacción natural, comportamiento, movilidad, experiencia compartida, etc.). [8.5]

Los dispositivos de control de los SRV (ver figura 1) se clasifican en sensores y efectores.

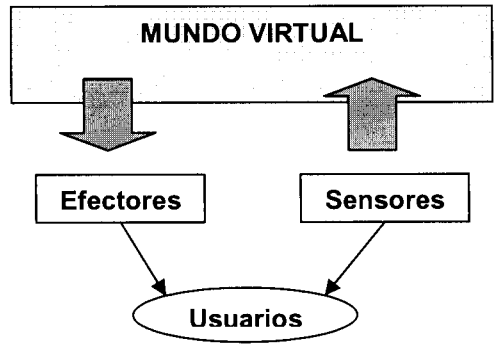

Fig. 1 Sistemas de Realidad Virtual. SRV

Dispositivos de entrada (sensores): se encargan de capturar el estado y las acciones del usuario y transmitir información hacia el entorno virtual. Pueden ser de localización (electromagnéticos, mecánicos, sónicos (ultrasonidos) y ópticos (rotoscopía)), y de control (ratones y joysticks3-D, bioeléctricos, electroguantes, etc.). [8.5]

Dispositivos de salida (efectores): son aquellos que estimulan los sentidos del usuario y transmiten información del estado del entorno virtual. Pueden ser visuales o foto-receptivos (visiocascos, sistemas binoculares, monitor estereoscópico + gafas, proyectores), auditivos, táctiles (mecánicos y térmicos), orientadores (mecánicos), olfativo-gustativos (químicos), y plataformas móviles. [8.5]

Atendiendo al modo de presentación visual, los Sistemas de Realidad Virtual pueden ser ***inmersivos*** (el usuario no se ve a sí mismo, y la estimulación es separada para cada ojo; es el caso típico de los cascos de RV); ***proyectivos*** (el mundo visual se proyecta en paredes, el usuario se ve a sí mismo y tiene la posibilidad de compartir experiencia multiusuario; en este caso se encuentra el

pilotado de vehículos); y ***de sobremesa*** (se trabaja a través de un monitor, por lo que el usuario se ve a sí mismo y pierde toda sensación de inmersión; este caso posibilita la alternancia entre el sistema de RV y otras aplicaciones). [8.5]

Los SRV varían según los niveles de inmersión como el ***WoW*** (*Window on a World*), donde se trabaja con el monitor tradicional y dispositivos como el ratón y *joysticks* (videojuegos); la ***TelePresencia*** o presencia simulada en entornos reales distantes, (Ej. TeleOperación de robots); y la ***Realidad Virtual*** (presencia simulada en entornos simulados, sensores/efectores usuario-aplicación. Ej. Simuladores de vuelo). [8.5]

Los SRV desarrollados actualmente por SIMPRO son sistemas proyectivos en entornos simulados, cuyos dispositivos de entradas son sensores de control que informan de cambios (como cambios en el timón de de un automóvil), y sensores de localización (como la localización por láser del punto en que "dio" un disparo de fusil en una pantalla donde se proyecta el campo de tiro), y los dispositivos de salida o efectores son de tipo visual, auditivo, y algunos orientadores (plataformas móviles).

1.2 Técnicas, algoritmos y tendencias actuales

A continuación se expone el estudio realizado que determina las técnicas y algoritmos a utilizar en este proyecto, dadas las tendencias para el desarrollo a nivel mundial de sistemas similares al que se está concibiendo en este trabajo.

1.2.1 Animación *Real-Time* contra *Single-Frame*

Las animaciones pueden generarse en tiempo real (*real-time*) o en modo de fotograma simple (*single-frame*).

La animación en ***tiempo real*** va generando los fotogramas a medida que son necesarios durante la propia corrida de la aplicación. Este "***rendering***" trabaja directamente con el *hardware* de la máquina, el cual debe ser capaz de soportar dicha tarea. [18.1]

Desde un punto de vista práctico, el tiempo real se consigue cuando el tiempo de respuesta del simulador es lo suficientemente corto como para que la percepción del fenómeno se corresponda con fidelidad al sistema real. Esto implica que las imágenes sean generadas a una velocidad tal que produzca la sensación de un movimiento persistente, normalmente tomada como 1/24 de un segundo; esta velocidad de generación (*framerate*) puede variar en dependencia de la fluidez de movimiento que se quiera lograr en una determinada aplicación, de los tipos de imágenes que se deseen mostrar y de las condiciones específicas en que se verán. [18.1]

Si las imágenes no pueden reproducirse a una velocidad suficiente como para proporcionar la animación de tiempo real (debido, por ejemplo, a un *hardware* insuficiente, o a que no se desea realmente obtener animaciones dinámicas), entonces puede simularse el *rendering* a nivel de *software* y no de *hardware*, lo cual lleva un proceso mucho más lento. Ésta animación en ***fotogramas simples***

o ***fotograma a fotograma*** consiste en generar previamente los fotogramas, almacenarlos y mostrarlos más tardes a velocidades suficientes como para crear un movimiento persistente. [18.1]

En los simuladores en que se requiere que las acciones de los objetos y personajes del escenario actúen como respuesta a señales del exterior a cada instante, no se pueden traer preparadas las animaciones correspondientes a las acciones (para hacerlo serían necesarias infinidad de combinaciones de animaciones que respondieran a todos las posibilidades del simulador, con la correspondiente saturación de la memoria de la máquina, lo cual no resulta nada apropiado), sino que hay que generar las animaciones dinámicamente y en un tiempo que para la percepción del ojo humano sea real. Esto sólo se puede lograr con la animación en tiempo-real. [18.1]. Se puede decir que el Tiempo-Real es la base de la Realidad Virtual. [8.5].

1.2.2 Modelado geométrico

Una escena puede contener distintos tipos de objetos (nubes, árboles, rocas, edificios, mobiliario, etc.), para los que existen una gran variedad de modelos de representación que se pueden clasificar en modelos alámbricos, de superficies de contorno (planas y curvas) y sólidos. [8.1]

Los ***modelos alámbricos*** son aquellos cuerpos que se representan a partir de las aristas que los componen. No tienen superficies, son los más simples, fáciles de construir, con pocas necesidades de memoria y almacenamiento, y se visualizan rápidamente, aunque tienen representación ambigua y falta de coherencia visual. [8.1]

Los ***modelos de superficies planas (poligonales)*** "son un conjunto de líneas rectas (arcos) que no se cruzan y que unen un conjunto coplanar de puntos (vértices) definiendo un área simple (habitualmente convexa y sin agujeros)"

[8.1]. La representación poligonal a través de triángulos mejora mucho las prestaciones del HW. Las tarjetas gráficas visualizan más triángulos/segundo de los que pueden enviarse por el *bus*. [8.1]

Los ***modelos de superficies curvas*** se definen por ecuaciones ***cuádricos*** (descritos mediante ecuaciones de segundo grado) y ***splines*** (representación por interpolación (la curva o superficie pasa por los puntos de control) o por aproximación (la curva se ajusta a los puntos de control)); existen otros tipos de ecuaciones para expresar el modelo a través de curvas y superficies por sólo citar algunas, ***Hermite***, ***Bezier***, y ***B-Splines.*** [8.1]

Los ***modelos sólidos*** tienen en cuenta además de las caracteísticas geométricas del cuerpo, los comportamientos físico mecánico de los mismos, por tanto son modelos más completos que los anteriores. Los modelos más simples son caracterizados a través de las ecuaciones cuadráticas y constituyen los cuerpos parametrizados tridimensionales que se pueden observar en cualquier *software* dedicado a la gráfica. A los cuerpos se les atribuye propiedades como masa, textura, color y se permite la interacción entre los mismos. [8.4]

Existen modelos no convencionales muy útiles en dependencia de cuánto realismo se le quiera dar al mundo virtual, como son: ***fractales*** (permiten representar objetos que por su complejidad no se ajustan a la geometría Euclídea como montañas, nubes, plantas, etc.); ***gramáticas*** (se parte de una forma simple y se va añadiendo nueva información geométrica aplicando una serie de reglas de transformación a la forma original. Ej.: simulación de la evolución de especies vegetales); ***blobs (burbujas o gotas)*** (esferas flexibles entre las que existe un campo de fuerza que las repele o atrae, de forma que cuando entran en contacto se funden en un único cuerpo con conexiones suaves; denominadas también *metaballs*, isosuperficies u objetos blandos, se aplican en el modelado de moléculas y representación de fluidos); y ***modelado basado en propiedades físicas*** (describen los objetos en términos de la

interacción con fuerzas externas e internas como la gravedad, útil para el modelado de telas y ropa, y en las animaciones para describir más exactamente trayectorias (ecuaciones dinámicas en lugar de cinemáticas)). [8.4]

Aunque la tendencia más simple para la representación de los efectos ambientales, es poner una animación correspondiente a un efecto, por ejemplo, fuego, con un fondo transparente (vía muy utilizada en los juegos actuales, pero inconveniente para manejar efectos como la lluvia), el manejo de los efectos ambientales es más general a través del uso de otro de los modelos no convencionales que es el ***sistema de partículas.*** Es ideal para objetos o fenómenos dinámicos (nubes, humo, fuego, explosiones,...), donde las partículas pueden tener distintas formas como puntos, líneas (ésta se usa a menudo para crear un efecto de rastreo, conteniendo la posición actual y última de la partícula), ***Texture-Maped Quads*** (esta variante ofrece una mayor flexibilidad, la partícula se representa como un cuadrángulo de dos triángulos sobre el que se dibuja una textura apropiada, por ejemplo, una imagen de una chispa), esferas, elipsoides, *blobs*, etc., y tienen características comunes, por ejemplo su tamaño o color, y un comportamiento que determina la evolución del sistema (tiempo de vida, aspecto de la partícula, trayectoria,...) y que puede estar influido por fuerzas específicas (gravedad, campos magnéticos,...). [8.4]

Modelado por niveles de detalles

Una técnica importante en la optimización del modelado de los cuerpos en la escena es el ***modelado por niveles de detalle (LOD).*** Consiste en que, según las distancias a las que se pueden encontrar los objetos respecto a la cámara de la escena, se optimizan los cuerpos conformándolos con la menor cantidad posible de polígonos, cantidad satisfactoria a su vez para la distancia a la que se encuentran. Normalmente estas diferentes representaciones de los objetos han sido generadas previamente. [8.4]

Modelado de terrenos

Una malla de un terreno no es más que una cuadrícula o red de triángulos. Pero con una altura por cada vértice que representará como se modela una transición suave de montañas a valles, simulando terrenos naturales. Esto por supuesto, lleva una textura adecuada. [3.2]

Para el manejo de los terrenos se utilizan los llamados ***Heightmaps*** o mapas de alturas, que no son más que arreglos donde cada elemento especifica la altura de un vértice en particular de la cuadrícula del terreno. Una variante del mapa de alturas es a través de un mapa de tonalidades de grises ***(grayscale map)***, donde los valores más oscuros representan los puntos bajos del terreno, y los más claros, los más altos. Esto brinda la posibilidad de almacenar un mayor rango de diferencias entre alturas. [3.2]

1.2.3 Transformaciones geométricas

Las transformaciones en 3D se utilizan para manipular objetos en el espacio 3D (traslación, giro y escalado) y visualizar y examinar objetos.

El primer paso para lograr las transformaciones geométricas es utilizar un ***sistema de coordenadas*** que defina el espacio de forma numérica y proporcione una métrica que permita describir la distancia entre dos puntos. Los sistemas de coordenadas pueden ser ***dextrógiro*** (el eje Z apunta hacia el exterior del papel; este es el utilizado para realizar las transformaciones geométricas) y ***levógiro*** (el eje Z apunta hacia el interior del papel). Se puede considerar la localización de los objetos con respecto a un punto central de referencia llamado origen. [8.2]

Representación matricial de las transformaciones

Todas las transformaciones son el resultado de una modificación de los valores de los vértices del cuerpo, dada por la multiplicación de los valores anteriores por un factor de modificación. Esta operación se puede representar utilizando matrices, donde se multiplica la matriz de transformación por la matriz columna de los vértices, obteniéndose la matriz columna de los nuevos vértices. Las transformaciones más comunes son: [8.2]

Traslación: Consiste en mover un objeto a una nueva posición. Las nuevas coordenadas vienen dadas por: x' = x +Tx, y' = y +Ty, z' = z +Tz.

Matriz:

$$\begin{vmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Escalado: Cambia el tamaño del objeto. Se realiza respecto a un punto. Si se realiza respecto al origen las nuevas coordenadas son: x'=x*Sx, y'=y*Sy, z'=z*Sz.

Matriz:

$$\begin{vmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Si |Sx|>1 y |Sy|>1 y |Sz|>1, aumenta el tamaño. Si |Sx|<1 y |Sy|<1 y |Sz|<1, disminuye. Si Sx=Sy=Sz, el escalado es uniforme, sino, no uniforme. Si Sx<0, el

objeto se refleja respecto al plano YZ. Si Sy<0, el objeto se refleja respecto al plano XZ. Si Sz<0, el objeto se refleja respecto al plano XY.

Reflexión: Refleja el objeto respecto a un plano. Es una extensión del escalado, con alguno de los parámetros Sx = -1, Sy = -1, Sz = -1.

Rotación: Se utiliza para orientar objetos. Se realiza respecto a un eje principal coordenado.

Matriz de rotación en X:

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Matriz de rotación en Y:

$$\begin{vmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Matriz de rotación en Z:

$$\begin{vmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Distorsión (shearing): Distorsiona la forma de un objeto. Se produce respecto de un eje.

Matriz:

$$\begin{vmatrix} 1 & b & c & 0 \\ e & 1 & g & 0 \\ i & j & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

La distorsión con respecto al eje X se controla con (b,c), respecto al Y con (e,g), respecto al Z con (i,,j).

Concatenación de transformaciones

Se pueden combinar varias transformaciones para obtener operaciones más complejas, a través de la obtención de una matriz como producto de la multiplicación de las otras. Como las matrices son cuadradas se obtiene una única matriz. A estas matrices se les suele llamar *SRT-transform* (escalar-rotar-trasladar). [8.2]

El producto de matrices no es conmutativo (M1*M2<>M2*M1), por lo tanto, la aplicación de transformaciones tampoco lo es. [8.2]

Transformaciones que son conmutativas (el orden no importa): [8.2]

- Traslación-Traslación.
- Escalado-Escalado.
- Rotación-Rotación.
- Escalado Uniforme-Rotación.

Transformaciones que no son conmutativas (el orden importa): [8.2]

- Traslación-Escalado.
- Traslación-Rotación.
- Escalado No Uniforme-Rotación.

(Una traslación después de una rotación no es lo mismo que la rotación después de la traslación).

Otro tipo de transformaciones es la llamada ***transformación no lineal***, que consiste en un conjunto de transformaciones que no se aplican de forma constante sobre todo el objeto. En lugar de utilizar constantes en las matrices de transformación se emplean funciones. A este tipo de transformaciones se les llama deformaciones globales. Entre las más conocidas se destacan: afilar (*Taper*), torcer (*Twist*) y curvar (*Bend*). [8.2]

1.2.4 Iluminación y colores

Las luces se utilizan para iluminar varios o todos los objetos en una escena. La combinación de luces, sombras y colores en la escena ayudan mucho en el realismo de un simulador.

Existen cuatro tipos básicos de luces que son:

Luz ambiental: es la luz de fondo que ilumina igualmente todas las superficies sin importar su posición u orientación. No viene de una dirección en particular. [7.4]

Luz direccional: fuente de luz que tiene dirección pero no posición finita. Simula, por ejemplo, los efectos de la luz solar. [7.4]

Luz punto: fuente de luz omni-direccional (radial) que tiene posición. Se atenúa con la distancia. [7.4]

Luz Spot: ilumina un área pequeña, con un cono de ángulo especificado (por ejemplo: linternas, luces de automóviles). Se atenúa con la distancia. [7.4]

Entre los atributos de las luces están los colores, que se utilizan para dar determinados efectos sobre los objetos iluminados, los tipos de colores de luces son:

Color de ambiente (ambient color): color de la luz que ilumina igualmente todas las superficies. [1.3]

Color de difusión (diffuse color): color de la luz que ilumina los polígonos como una función del ángulo entre la dirección de la luz y la normal del polígono (o entre la dirección de la luz y las normales de los vértices, en el caso de que se aplique *gouraud shading*). [1.3]

Color especular (specular color): color que afecta la intensidad que es reflejada por una superficie brillante (*shiny surface*). [1.3]

Materiales

Un material, desde el punto de vista de expresión gráfica, es la combinación de luces y colores usados para definir una apariencia. Los cuerpos pueden emitir luz, reflejarla, o ambas cosas, efectos que se manifiestan a través de colores. Al diseñar un cuerpo se consideran dos tipos de colores: [7.3]

- Colores usados en el cuerpo como tal.
- Colores de la luz que actúa sobre el cuerpo.

Colores

Los colores en los *softwares* gráficos 3D generalmente son almacenados como colores **RGB** (*Red Green Blue*). Los colores RGB consisten en tres números reales, que representan los niveles de rojo, verde y azul, respectivamente. Los

programas de *rendering* dan sus salidas en colores RGB, y usan colores RGB para calcular la mezcla y filtrado de los colores de las luces. [2.1]

El rojo, el verde y el azul se conocen como colores aditivos primarios porque cualquier color de luz puede ser representado mezclando estos tres colores. Cuando se combinan en igual proporción en el *rendering*, forman una luz blanca. [2.1]

Colores de vértices

Se pueden usar los colores de vértices para incrementar el realismo visual en una escena virtual, almacenándose información de luces, sombras y reflexiones en los vértices de la malla, con los que se hace una interpolación entre los vértices produciendo un efecto de alisamiento. [7.2]

Los colores de vértices pueden representar además otras magnitudes físicas como temperatura o presión. [7.2]

1.2.5 Texturizado

Texturizar es aplicar una textura (una imagen) a un modelo de manera que le sirva de "piel", incrementando los detalles y el realismo de la escena significativamente. [3.1]

La principal técnica de texturizado es el ***texture mapping o mapeo de texturas,*** la cual se basa en una correspondencia entre los vértices de la malla de los cuerpos y los puntos en la textura. El par (u,v) del mapa de texturas identifica un elemento de la textura llamado ***texel***. La axisa **v** crece a medida que se baja en el mapa. Ambas axisas cubren el rango [0, 1], lo que permite trabajar con texturas de cualquier tamaño. Por cada triángulo 3D, se define un triángulo correspondiente en la textura y esta correspondencia se almacena en el *mapa de texturas.* [3.1]

1.2.6 Proyecciones

Aunque el mundo virtual creado sea en tridimensional, los visores actuales son *displays* bidimensionales, por lo que se utilizan las proyecciones para convertir las coordenadas 3D del mundo a las coordenadas 2D de la pantalla de un dispositivo.

En materia de gráficos 3D se manejan dos tipos de proyecciones:

La primera, llamada ***proyección paralela (u ortogonal)***, es aquella en que todos los objetos mantienen sus dimensiones en la proyección plana sin importar cuan lejos estén en el mundo. [1.3]

La otra, llamada ***proyección perspectiva***, determina los tamaños de los objetos basándose en la distancia de los objetos al plano de proyección (lente de la cámara). Con esta proyección, todos los rayos de luz que rebotan de los objetos convergen en un simple punto para el ojo del observador. [1.3]

1.2.7 Tratamiento de colisiones

La detección y respuesta ante colisiones permiten determinar cómo un objeto en el mundo virtual interactúa con los demás.

Uno de los acercamientos más sencillos y directos a la detección de colisiones, es usando los objetos ***Bounding Spheres o esferas fronteras*** (objetos ocultos que bordean a cada objeto del mundo). En dependencia de la forma del objeto, se calcula el radio de la esfera frontera, que coincide con la distancia del centro al vértice más alejado de éste. Para comprobar las colisiones, se chequea si la distancia entre los centros de dos objetos es menor que la suma de los radios de sus dos esferas fronteras. [1.5]

Sin embargo, cuando se tiene un objeto irregular, a menudo la esfera frontera ocupa un gran espacio en el que el objeto realmente no existe, lo que da una colisión irreal en esta parte. Para resolver este problema, se emplean entonces los ***Bounding Boxes o cajas fronteras***, que se determinan buscando los mayores y menores vértices en relación con el centro del objeto, hallándose una caja que contiene al objeto, alineada a las axisas. [1.5]

Pero esta técnica no es tampoco exacta en todos los casos. En dependencia de la aplicación que se esté haciendo, se pueden usar ambas técnicas como pruebas iniciales. Se puede usar una jerarquía de pequeñas esferas y cajas en el objeto. [1.5]

Otros objetos fronteras utilizados son: cápsulas, *lozenges* (pastilla, gragea), cilindros y elipsoides. [1.5]

1.2.8 Visibilidad

El proceso de *rendering* de la escena hasta la imagen final se produce, en los sistemas en tiempo real, mediante una serie de fases consecutivas y conectadas entre sí (la salida de datos de una fase constituye la entrada de la siguiente), que se denomina ***pipeline (tubería) gráfica.*** En dicho proceso gráfico, los datos necesarios deben transmitirse de una fase a otra, lo que en algunos casos puede ser más lento que el propio procesamiento de los datos, especialmente cuando se utiliza el *bus* del sistema. Por tanto, hay que minimizar la cantidad de polígonos que se envían al proceso de *rendering*, o sea, enviar solamente lo que es realmente visible. [14.1]

La determinación de visibilidad consiste en decidir qué partes del mundo están visibles desde una localización. Existen varias técnicas y algoritmos para la manipulación de los objetos del mundo virtual, que optimizan la animación o

rendering de la escena garantizando una gran velocidad de ejecución de los simuladores. Algunos de ellos se exponen a continuación.

El volumen de visión

La primera idea para optimizar la visualización de cuerpos es representar solamente aquellos que están frente a la cámara de la escena. La solución más común para estos casos es crear un *view volume* o volumen de visión, una sección de espacio 3D visible por la cámara (***Frustum***). Cualquier cosa fuera del volumen de visión no será vista por la cámara, y por tanto no se tendrá en cuanta en la proyección. En la proyección perspectiva, el volumen de visión tiene forma de pirámide. [1.3]

La idea es determinar qué objetos se encuentran en el volumen de visión de la cámara y sólo estos serán enviados, de primera instancia, al proceso de *rendering*.

Organización de los objetos para el control de su información y visibilidad. El grafo de la escena

Para lograr un *rendering* eficiente de la escena, primeramente se necesita tener el control de los objetos que en ella se encuentran. Existen dos variantes para organizar los objetos: [8.3]

Variante 1: agrupar los objetos en una lista e iterar para escogerlos y aplicarles el *rendering*. Esto no es eficiente si cada objeto dibujable debe ser revisado.

Variante 2: agrupar los objetos jerárquicamente de acuerdo a su localización espacial.

La segunda es obviamente más eficiente, y no es más que representar en un grafo toda la información manejable de la escena, tanto los objetos dibujables y no dibujables, como sus características. [8.3]

Generalmente, un grafo de escena presenta nodos intermedios (que contienen información para aplicar *render* a los objetos como posición, orientación y escala, volúmenes fronteras (utilizados para el *culling* jerárquico y la detección de intersecciones), estado de *render*...), y nodos hojas (objetos en cuestión). [8.3]

Por ejemplo, si se quisiera representar un auto en escena, algunas hojas del grafo se corresponderían con partes del auto como las ruedas, como objetos separados, y el auto sería un nodo intermedio que agruparía a los nodos hojas (sus partes). El nodo-grupo "auto" contendría información común a todas sus partes, y los estados que lo modifiquen modificarán a todas sus partes.

En el grafo de la escena, las transformaciones pueden ser locales o globales. Las locales son las que ocurren sobre un único objeto, es decir, sobre una de las hojas del árbol, (en el caso del auto sería una de las ruedas), y las globales las que ocurren sobre un grupo de objetos (se le aplica la transformación a todas las partes del auto). [8.3]

Al atravesar una rama y llegar a un nodo hoja, se tiene toda la información necesaria para el *rendering*.

Una variante de organización de los objetos de la escena es mediante un grafo acíclico, donde un mismo nodo puede tener varios padres, lo que ahorra espacio de almacenamiento pues si se quiere dibujar varias copias del mismo objeto, se coloca como hijo de varios nodos, con lo cual el algoritmo de recorrido llegará hasta él varias veces, por caminos diferentes y se dibujarán varias copias. [14.2]

En general, el grafo de la escena facilita: [7.1]

- Segmentar en pequeñas piezas la escena (subárboles). Esto permite aplicarle efectos a una determinada parte de la escena independientemente, así como una eficiente transmisión de efectos.
- Localización espacial, útil en la eliminación de regiones, minimizándose la cantidad de datos enviados al *render*. Acelera además la detección de colisiones ("detección jerárquica").
- Organización de los objetos en una jerarquía espacial y semánticamente, especialmente los caracteres humanoides.
- Salvar el estado del simulador con facilidad, salvándose cada nodo recursivamente.

Uso de Jerarquías: "Usar las jerarquías consiste en explotar la coherencia del objeto (partes de un objeto no intersectan si el objeto no lo hace)". [8.3] En el ejemplo de la figura, si el edificio no intersecta, sus plantas y habitaciones no lo hacen tampoco.

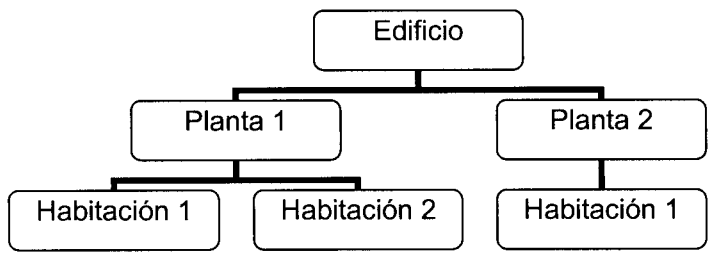

Fig. 2 Ejemplo de jerarquía.

En un grafo de escena se puede aprovechar la jerarquía de objetos sobre todo para determinar las colisiones de los objetos. Con esto se determina no sólo qué objetos se intersectan entre sí, si no cuáles colisionan con el campo de visión de la cámara, con lo que se conoce si el objeto está visible y si es necesario ser dibujado.

Merging (fusión, unión) de fronteras: a través de los grafos de escenas es posible crear nodos padres que agrupan a otros nodos, por ejemplo, el *grouping node* "auto" antes mencionado, que contiene a los nodos ruedas, etc. Cada una de sus partes generalmente tiene volúmenes de colisión independientes, pero toma mucho tiempo revisar cada una de ellas para detectar colisiones, cuando el auto entero pudiera estar fuera de colisiones. Para esto se utiliza el *bound merging* o combinación de fronteras, que no es más que crear un objeto frontera para colisiones que contenga a los objetos fronteras de sus hijos. Cuando la frontera combinada no colisiona no es necesario continuar la búsqueda por sus partes. [8.3]

Niveles de detalles (LOD): cuando se han modelado los objetos para distintos niveles de detalles, es posible entonces hacer un control del nivel de detalle (ajuste del LOD óptimo), y visualizar los cuerpos en dependencia de la distancia a la que se encuentran a la cámara. A través del grafo de la escena se pueden identificar para un mismo objeto sus LOD, por lo que durante el recorrido se puede escoger cuál de sus versiones representar. [8.3]

Organización de los objetos por profundidad.

Una vez que se ha determinado qué objetos están en el volumen de visión, es conveniente determinar cuales están realmente visibles y, dentro de ellos, cuales quedan directamente visibles o no. Para esto se suelen organizar por profundidad en árboles binarios (*BSP-Trees*), y se les aplican algoritmos de oclusión que determinan qué objetos son ocultados por otros para no ser enviados al *rendering* innecesariamente. [8.3]

Para una primera versión de este proyecto solamente se determinará qué objetos están en el volumen de visión de la cámara, y de aquí serán enviados al proceso de *rendering* aprovechando las facilidades de ordenamiento por *Z-buffering* y eliminación de caras ocultas que brinda el *hardware*.

Depth buffering o Z-Buffering:

Z-Buffer es una memoria de profundidad cuya existencia es una característica particular de los sistemas de visualización en tiempo real. [14.1]

Z-Buffer trabaja sólo con los polígonos interiores al prisma de visión. En el Z-buffer o *buffer* (espacio de memoria) de profundidad en Z, se almacena la profundidad medida entre los planos cercano y lejano del *frustum* de la cámara, visualizándose el "píxel" más próximo al ojo (aquel cuya componente Z en el sistema de la cámara tenga un valor mayor). Un píxel es dibujado sólo si su profundidad indica que está frente a un píxel previamente dibujado. [8.3]

Z-Buffer no precisa ordenamiento de polígonos, maneja todo tipo de escenas y evita cálculos inútiles de sombreado. No es muy óptimo para *software* pero para aceleradores de HW resulta muy bueno. [8.3]

Eliminación de caras traseras (Back face rejection): consiste en eliminar el *rendering* de los polígonos que no están directamente expuestos a la cámara, aplicable a objetos poliédricos cerrados (en especial: poliedros convexos). Da una reducción media del 50% de polígonos incrementándose significativamente la velocidad. [7.2]

La cara frontal de un polígono es aquella cuyos vértices están ordenados en sentido antihorario. [7.2]

La eliminación de caras traseras también se puede controlar a través de diversas técnicas como el BSP, Octree, sistemas de portales, paginado de terrenos, y otras. [7.2]

1.2.9 Animación

La animación, es la simulación de un movimiento creada por la muestra de una serie de imágenes o fotogramas. [11.1]

Es necesario conocer la diferencia entre video y animación: el video toma el movimiento continuo y lo descompone en fotogramas, la animación parte de varias imágenes estáticas y las une para crear la ilusión de un movimiento continuo. Mientras más imágenes consecutivas conformen una animación, ésta será mucho más real. Lo que distingue a la animación de las técnicas de adquisición de imágenes en movimiento, es que en la animación los fotogramas son generados uno a uno, bien por métodos tradicionales de dibujado, bien generando las imágenes en un ordenador, es decir, cada fotograma es generado de modo independiente. [11.1]

Fenómeno fisiológico base de la animación

La animación descansa en el fenómeno fisiológico de la persistencia de la visión. Los estímulos de una imagen permanecen en la retina entre 100 y 200 milisegundos. Si las imágenes que estimulan la retina son muy parecidas entre sí, podemos aprovechar este fenómeno para generar la percepción de imágenes en movimiento. La constatación de la persistencia de la visión data de 1824, año en el cual Peter Roget presentó ante la *British Royal Society* su trabajo titulado "*The persistence of vision with regard to moving objects*". [12.1]

Principios básicos de la animación

La animación tradicional cuenta con muchos años de experiencia y durante estos años, que también han sido de experimentación, se han hallado los siguientes principios básicos que han sido aplicados a la animación de personajes tridimensionales por ordenadores, y que les da un mayor realismo:

Difuminado del movimiento: Si las imágenes no son refrescadas con la suficiente velocidad se produce lo que se conoce como efecto ***estroboscópico***, las imágenes parecen que se suceden a saltos. Para evitar este fenómeno cada fotograma es una interpolación de una imagen y la siguiente. [12.1]

Anticipación: Preparación de la siguiente acción (por ejemplo, antes de salir corriendo, forzar una postura en el sentido contrario). La anticipación permite preparar los músculos para la acción, preparar al espectador y captar la atención hacia la acción principal e indicar la velocidad de la acción. [12.1]

Estiramiento y compresión: Para conseguir movimientos fluidos y sensación de elasticidad se usan deformaciones como el estiramiento y la compresión. Los objetos se deforman en la dirección de desplazamiento para dar sensación de peso y gravedad. La deformación es perpendicular a la trayectoria en los impactos. La regla básica consiste en mantener el volumen de los objetos constante. Con estas deformaciones se evita el efecto estroboscópico de forma semejante al difuminado por movimiento. [12.1]

Solapamiento y continuación de las acciones: El solapamiento consiste en comenzar la siguiente acción antes de terminar la anterior (por ejemplo, para abrir una puerta, el personaje al acercarse va estirando la mano antes de llegar). La continuación significa que los movimientos no se detienen bruscamente, sino que continúan más allá de su posición final (por ejemplo, al golpear un pelota con una raqueta esta continúa su movimiento por inercia mucho después de haber golpeado a la pelota). [12.1]

Las emociones

La personalidad de los actores digitales es transmitida a través de las emociones, y las emociones son el mejor indicador de cuán rápida debe ser una acción. Un personaje pudiera no hacer una acción particular de la misma manera en dos estados emocionales diferentes. Cuando un personaje está feliz,

sus movimientos serán más rápidos. Por lo contrario, cuando está triste, los movimientos serán más lentos. [19]

Para hacer que un personaje parezca más real, debe ser diferente de los demás personajes. Una manera simple de distinguir personalidades es con el contraste de los movimientos.

1.2.9.1 Animación por ordenadores

Se puede definir la animación por ordenador como la "generación, almacenamiento y presentación de imágenes que en sucesión rápida producen sensación de movimiento", un formato de presentación de información digital en movimiento a través de una secuencia de imágenes o fotogramas creadas o generadas por la computadora. [11.1]

La animación por ordenador permite representar modelos que evolucionan a lo largo del tiempo (no sólo cambian su posición, sino quizá también sú tamaño, color, iluminación, textura...).

1.2.9.2 Animaciones 3D por ordenadores

Una característica importante de la animación por computadora es que permite crear escenas "realmente" tridimensionales. Esto quiere decir que, a diferencia de la animación dibujada a mano, en una escena animada por computadora es posible cambiar el ángulo de la cámara y con esto, ver otra parte de la escena. [11.1]

Con la animación por computadoras se pueden reutilizar partes de la animación por separado, una animación puede verse muy diferente simplemente cambiando el ángulo de la cámara o el tiempo del movimiento de partes de la animación. Por ejemplo, si se tiene un conjunto de elementos iguales que se

mueven de forma similar, se puede hacer que cada uno de ellos tenga ligeros movimientos independientes además del movimiento que hace el grupo. De esta manera la animación se ve mucho más dinámica. [11.1]

Es posible lograr que una animación se vea más realista si se varía el peso y el tamaño de los objetos. Gracias a las nuevas técnicas de graficación, los objetos se pueden ver mucho más realistas. Se puede incluso hacer que aparenten ser de un material específico cambiando las texturas y los pesos. Entonces, al hacer animaciones, se deben considerar los atributos que tiene cada elemento de la animación. [11.1]

También se debe tener en cuenta en las animaciones 3D la inercia de los cuerpos, cómo interactúan los cuerpos al colisionar con otros, etc.

1.2.9.3 Animación de personajes 3D por ordenadores

La animación 3D, igual que el diseño de gráficos 3D, es mucho más compleja que la bidimensional, y requiere por lo general de una gran potencia de cálculo para ser elaborada con calidad, y un elevado tiempo de diseño para producir efectos realistas de movimiento, especialmente en lo que respecta a la animación de personajes o a la generación de entornos. [10]

La animación de personajes en 3D normalmente implica la definición de los distintos segmentos tridimensionales y la unión entre ellos, definiendo puntos de conexión y puntos de rotación que permitirán hacer la animación. Posteriormente habrá que definir cuáles son los *frames* que interesan y dibujar por separado cada uno de ellos, formando una secuencia de imágenes realistas que finalmente se unirán secuencialmente en la composición. [10]

1.2.9.4 Las leyes de la Naturaleza

La animación tradicional a menudo rompe las leyes de la Naturaleza, y suele definir movimientos atractivos y con carácter, pero imposibles en la realidad. Para realizar animaciones realistas, hay que tener en cuenta esas leyes de la Naturaleza: animación basada en leyes físicas, que utiliza la cinemática y la dinámica. Muchos movimientos cotidianos son muy difíciles de reproducir. [12.1]

La cinemática estudia los movimientos con independencia de las fuerzas que los producen, y se usa en animación en dos variantes:

La cinemática directa (direct kinematics): es la posibilidad de mover algunas de las "piezas" de un personaje o montaje 3D actuando sobre un punto y produciendo un movimiento sobre su eje o centro de rotación (por ejemplo, mover el brazo fijada la rotación sobre el hombro. El programa de animación 3D genera con fórmulas geométricas simples todos los movimientos necesarios de las partes ligadas a su vez a ella. En este caso, en la jerarquía de movimientos o giros definida, se parte de un eje más importante fijo (por ejemplo, el hombro) para mover elementos más sencillos (por ejemplo, el brazo). [12.1]

La cinemática inversa (inverse kinematics): es la posibilidad de que, moviendo elementos más sencillos en la jerarquía, el programa interpola el resto de articulaciones o puntos de giro, que pueden ser configurados por el animador, para conseguir que se muevan acorde a eso. Este tipo de movimiento es mucho más interesante pero a la vez más complejo, ya que en general no hay un sólo modo de rotar los elementos entre sí para conseguir seguir el movimiento final que pretende el usuario. Por ejemplo, un codo puede girar en un sentido, pero no en otro. Por ello pueden configurarse márgenes de rotación que indiquen al *software* qué límites tiene a la hora de elegir entre unos movimientos u otros. [12.1]

La dinámica estudia el movimiento teniendo en cuenta las fuerzas que lo producen. Se puede obtener gran realismo, pero resulta difícil especificar la

animación. Hay que tomar en consideración masas, aceleraciones, grados de libertad, restricciones al movimiento, movimientos prioritarios... La dinámica de los cuerpos rígidos articulados es más sencilla que la de los cuerpos deformables. Se distingue:

Dinámica directa: a partir de las masas y fuerzas aplicadas, se calculan las aceleraciones. [12.1]

Dinámica inversa: a partir de las masas y aceleraciones, se calculan las fuerzas que hay que aplicar. [12.1]

Otras técnicas utilizan curvas tridimensionales flexibles o algunas otras variantes basadas en polígonos en lugar de esferas. Técnicas más avanzadas de animación 3D emplean otros enfoques radicalmente distintos, como deformaciones, *morphing* (técnica de animación por la cual una imagen es gradualmente convertida en otra), sistemas de partículas, basados en simulación de fenómenos naturales, etcétera. [12.1]

1.2.10 Técnicas de animación de personajes 3D

Animación basada en fotogramas: La animación basada en fotogramas es una de las más utilizadas. Para hacer una secuencia, se van filmando las imágenes fotograma por fotograma y luego estos se unen para formar la animación. Es posible formar bibliotecas de movimientos de cada parte del cuerpo de la animación para de esta forma combinarlas y hacer animaciones diferentes. [11.1]

KeyFramming: El *keyframming* se refiere a establecer posiciones en puntos específicos de tiempo en una animación y la parte intermedia la obtiene la

computadora por medio de la interpolación matemática. Es necesario hacer un *keyframming* para cada control en cada nivel de la jerarquía del modelo. [11.1]

Esta técnica conocida también como "animación por cotas" consiste en basar el movimiento en unos fotogramas fundamentales o claves ("*keyframes*") y luego dejar que el sistema genere automáticamente los fotogramas intermedios mediante métodos de interpolación. Es importante que las cotas sean representativas del movimiento para que la interpolación tenga suficiente información. Esta técnica está basada en los métodos de trabajo de la animación tradicional en la que los animadores más expertos dibujan los momentos fundamentales del movimiento (cotas o *keyframes*) y los animadores principiantes dibujan los fotogramas intermedios ("*inbetweens*"). [11.1]

Rotoscopiado: El rotoscopiado consiste en una forma más elaborada de *keyframming,* donde se captura un movimiento real, y se utiliza esa información para mover un diseño generado por ordenador. [10]

Wavelets: *Wavelets* significa "pequeñas ondulaciones". Esta técnica permita que en una sola imagen se compriman una gran cantidad de datos para que al acercarse a ellas, se vallan viendo los detalles sin distorsión. [10]

Animación procedural: consiste en describir el movimiento de forma algorítmica. Hay una serie de reglas que controlan como se van modificando los distintos parámetros (como la posición o la forma) a lo largo del tiempo. Para movimientos sencillos (un péndulo o una rueda que gira) es una buena solución, pero para movimientos más complejos (una persona caminando, o una moneda que cae al suelo), resulta difícil conseguir buenos resultados. Hay algunas técnicas con resultados interesantes, como los sistemas de partículas o la simulación de movimientos grupales. [10]

Animación por esqueleto: consiste en deformar la malla de un cuerpo a través de una estructura jerárquica de huesos, a los que se le asocia un grupo de

vértices. Es una técnica sumamente utilizada actualmente por los programadores, dada su rapidez para procesar y obtener excelentes resultados, y la posibilidad de incluir una gran cantidad de detalles en la animación de un personaje, desde las arrugas de la piel hasta la definición de los músculos de su cuerpo. [4.1]

1.2.10.1 Sistema de animación por esqueleto

Con el sistema de animación por esqueleto, para representar a un personaje y su animación, se necesitan junto a los datos de una malla 3D, una jerarquía de "huesos". Estos últimos son una serie de transformaciones jerárquicas que, como los huesos en el cuerpo humano, permiten la deformación de este. Usualmente, los datos huesos son representados básicamente como matrices de transformación. Estos eliminan la necesidad de almacenar la posición de los vértices por cada *frame* de la animación, superando ya desde este punto a la animación basada en vértices. Otras ventajas de la animación por esqueleto está en la suavidad del cambio de una animación a otra, además, como las animaciones son independientes a las mallas, en una escena una misma animación puede aplicársele a diferentes mallas. [4.1]

La deformación puede ocurrir dado que cada hueso según su posición lógica con respecto a la mallas tiene influencia sobre una serie de vértices de la misma. Las últimas dos maneras que surgiendo con el tiempo para mejorar la técnica de deformación que dan lugar a lo que podemos ver actualmente en los videojuegos y simuladores más avanzados del mundo son las que siguen, enunciadas en orden de surgimiento: [20]

Stitching: Con esta técnica los vértices correspondientes a cada hueso son transformados simplemente aplicándoles la matriz de transformación del hueso para cada *frame* a mostrar. La deficiencia que esta técnica es que pierde suavidad la malla en las intersecciones de los huesos pues al flexionarse una

articulación del cuerpo los vértices más cercanos a esta y que pertenecían a huesos diferentes se alejan demasiado, lo que crea ángulos abruptos en la zona de la articulación y una sensación de rigidez. [20]

Skinning: Esta técnica es básicamente como la anterior pero soluciona su deficiencia. Los vértices pueden ser influenciados por más de un hueso. Para estos casos cada hueso tiene un valor entre 0 y 1 llamado *weight* (peso) que indica cuanta influencia tiene este sobre el vértice. La sumatoria de los pesos de diferentes huesos para un vértice es igual a 1. En este caso para obtener un vértice transformado sería: [20]

VérticeFinal = Vértice * Matriz1 * Peso1 + Vértice * Matriz2 * Peso2 + ...
+ Vértice * MatrizN * PesoN

Donde las matrices y los pesos pertenecen a cada hueso con influencia sobre el vértice. [9]

Jerarquía de personajes.

La jerarquía de un personaje muestra la forma en que van estar enlazadas las diferentes partes del cuerpo de un personaje. Cuando se manipula una jerarquía, moviéndose al elemento padre se mueven todos sus hijos. Esto también es conocido como cinemática directa. En la cinemática inversa sucede lo contrario, los elementos hijos manejan a los padres.

La jerarquía para el cuerpo de un personaje es similar a un árbol, parte de un tronco simple y este controla todas las ramas. En un esqueleto humano, el tronco siempre es la cadera. Esto se debe a que es aproximadamente el centro de gravedad del cuerpo humano y el centro de la distribución del peso de todo el cuerpo. [4.1]

La cadera soporta la columna y toda la parte superior del cuerpo humano, pasándole el peso hacia abajo a las piernas. Esto hace que la columna y las

piernas sean hijos de las caderas. Además, la columna es padre de los hombros que a su vez tienen a los brazos como sus hijos, y las piernas padres de los pies. A continuación se muestra gráficamente esta explicación.

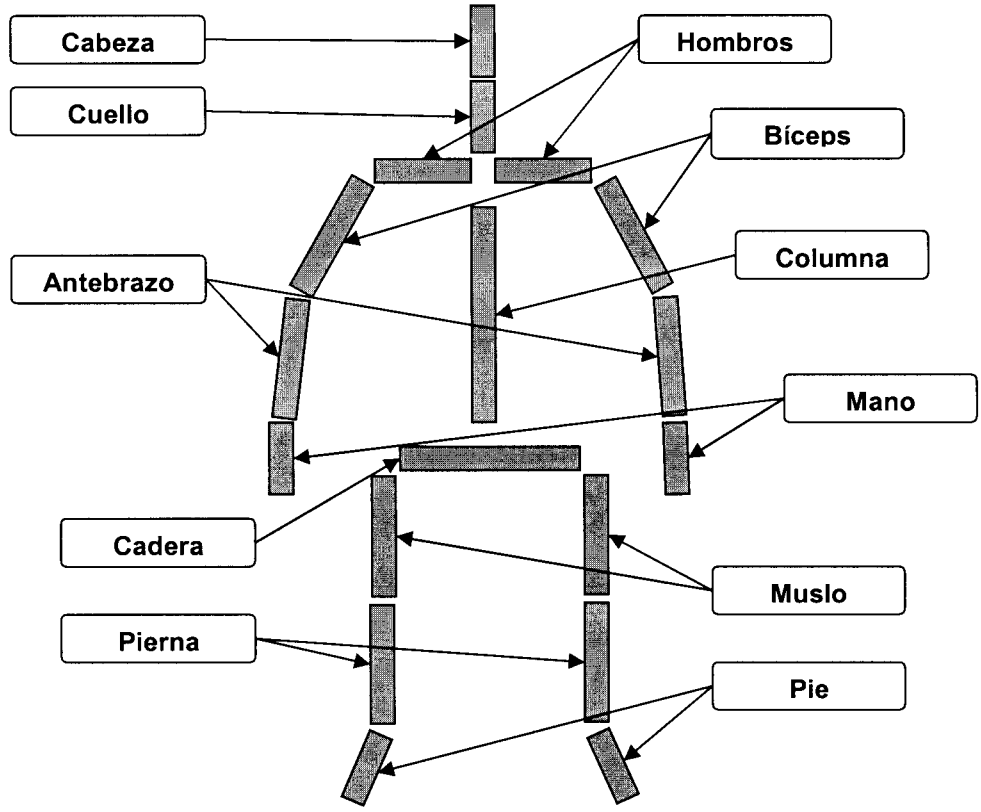

Fig. 3 Jerarquía básica de un cuerpo humano.

Manipulación de la jerarquía

Una forma de manipular una jerarquía es que al moverse el elemento padre se muevan todos sus hijos. Esto también es conocido como cinemática directa. Otra manera es llamada cinemática inversa, donde sucede lo contrario, los elementos hijos manejan a los padres. Ambas tienen ventajas y desventajas según donde sean utilizadas.

Para la animación de personajes la cinemática inversa tiene algunas más opciones que la cinemática directa pero esto también la hace más compleja. Sin

embargo, después de establecida hace mucho más rápida la creación de una animación. La animación inversa por otro lado, es muy sencilla de utilizar. Decidir cual técnica utilizar depende de que parte del esqueleto se quiere animar. [4.1]

La cinemática directa es más comúnmente usada en la parte superior del cuerpo: la columna, los hombros y los brazos. Si el personaje, por ejemplo, está de pie y gesticulando con las manos, con esta técnica se puede hacer muy buen trabajo. Por otro lado si básicamente el personaje realiza una acción que involucra tener contacto con alguna otra cosa, ya sea objeto u otro personaje debe ser utilizada la cinemática inversa. La columna y las manos son las más usualmente animadas con cinemática directa. [4.1]

La cinemática inversa siempre es usada para la animación de las piernas y los pies. Esta es ideal para la acción de colocar los pies del personaje sobre el suelo. Si el personaje está nadando, sin embargo la cinemática directa es más útil tanto para piernas y pies como para los brazos. [4.1]

1.2.11 Ficheros de modelo y animación.

El fichero que el cliente estableció para cargar los modelos de mallas es el ***RTX***, una versión avanzada del formato ***ASE*** (ASCII *Scene Export*). Esto le permite heredar su fácil interpretación y por lo tanto su fácil implementación. Además soporta, a diferencia del formato ASE, lo más actual del modelado de objetos mallados: el *Skinning*. [21]

Este fichero permite almacenar, para un objeto geométrico, la escala en que fue diseñado, su lista de vértices (coordenadas x, y, z) y caras (trío de vértices que conforman un triángulo de la malla). También permite almacenar la textura del objeto y por consiguiente la lista de coordenadas de textura (u,v) correspondiente a cada uno de los vértices de las caras de la geometría. [21]

Para el caso de los modelos de malla que contengan esqueleto, este fichero almacena la información de cada uno de los huesos. Estos se encuentran dispuestos en el fichero consecutivamente. En el espacio que le corresponde a cada uno de estos elementos se almacena su nombre, su índice, el índice del padre, su posición, rotación y escala para el sistema de coordenadas locales, su posición con respecto al padre, y la lista de vértices sobre los cuales tiene influencia, con los pesos correspondientes a cada uno de esos vértices. [21]

También, el cliente ha establecido que el formato de fichero para la carga de las animaciones será el ***BVH*** (*BioViosion Hierarchy*), de captura de movimientos, de la compañía BioVision. Es un fichero en formato texto, muy conocido en el mundo de la animación por ordenadores. [16]

La principal razón para la elección de este formato de fichero, es que brinda toda la información necesaria para animar cualquier personaje que tenga la misma jerarquía de huesos, independientemente al diseño de su malla. Ésto reduce el número de animaciones a cargar cuando se tienen varios modelos con una misma jerarquía de huesos. Además, el cliente está interesado en que las animaciones sean a partir de captura de movimientos dado el realismo que le imprime a la animación.

Este fichero está compuesto por dos partes en lo que se refiere a tipos de información. La primera es conocida como la parte jerárquica (*HIERARCHY*), en donde se encuentra la información concerniente a la jerarquía del esqueleto al que se le aplica la animación. Contiene la posición relativa de los huesos definida como "*Offset*", además a través de la palabra "*Joint*" se van enlazando los huesos de padre a hijos hasta conformar todo el esqueleto. [16]

Es importante aclarar que a pesar de que el fichero BVH contiene la jerarquía del esqueleto, este no almacena la información de la escala ya que ésta depende del diseño de la malla que lo usará, y se define en el fichero que contiene el modelo de malla con huesos, el cual se tratará más adelante. El

escalado también puede modificarse a decisión del programador que utilice el modelo. [16]

La otra parte que compone al fichero BVH, es la sección de movimiento (*MOTION*). En ella se guardan datos de la cantidad de *frames* que tiene la animación, el tiempo de duración de un cuadro en pantalla o *Frame Time* (en segundos) y por útimo, se encuentran los datos de la animación. Estos se distribuyen en una serie de filas y columnas que contienen los valores de la rotación de los huesos en los tres ejes de coordenadas. Las tres primeras columnas son los valores de las coordenadas de posición del hueso raíz (por ejemplo, la cadera para un humanoide), y el resto son las rotaciones por ejes de cada hueso del esqueleto. El orden es el mismo en que aparecen los huesos en la sección *HIERARCHY.* Las filas representan los *frames,* por lo que existirán tantas filas como *frames* tenga la animación almacenada. [16]

1.2.12 El Tiempo

El tiempo es una característica o atributo del mundo real, por tanto, si se quiere hacer un modelo que simule lo más fielmente posible la realidad, hay que mantener un rastro del tiempo en el mundo virtual lo mejor posible.

Pero existe un problema, y es que el mismo simulador, con exactamente el mismo código, e incluso usando el mismo ejecutable, puede correr mucho más lento o más rápido dependiendo de la computadora en que se ejecute. [22.2]

Para solucionar esto, es necesario "forzar" a que el simulador corra siempre de la misma forma independientemente del CPU. Existen dos formas de hacer este sincronismo: en base al *framerate* (razón o frecuencia en que se muestra la animación, número de frames por segundo), y en base al tiempo. [22.2]

La ***sincronización en base al framerate*** consiste en detener el ciclo principal de simulación hasta que se desee ejecutar el siguiente ciclo. Por ejemplo, si se

quiere que el simulador corra a 24 fps., se fija el *framerate* y se fuerza a que cada ciclo dure 0.041 segundos. Esto se logra chequeando al final de cada ciclo si se consumió el tiempo asignado para éste, y si no ha terminado se espera hasta que se cumpla dicho tiempo. De esta manera no importa la velocidad del procesador, pues siempre se dibujarán la misma cantidad de *frames* por segundo. [22.2]

Éste método tiene como desventaja, que cuando se corra la aplicación en una máquina muy lenta, el ciclo podrá demorar más que el tiempo asignado para un *frame*, e igual, no existirá el sincronismo deseado. [22.2]

En cambio, en la ***sincronización en base al tiempo*** no importa el *framerate* que tenga el simulador, de hecho ésta es su gran ventaja. Consiste en hacer el ciclo sin espera intermedia, y manteniendo el intervalo de tiempo transcurrido entre el ciclo anterior y el actual; con este tiempo, se calculan las nuevas posiciones de los objetos, tan sencillo como: si transcurrió poco tiempo, se moverán menos, si transcurrió un mayor tiempo, se moverán más. Es decir, hay que ajustar las distancias según el tiempo del ciclo. [22.2]

Pero esto tampoco significa que sea infalible: en una computadora muy lenta, a pesar de que los objetos se sigan moviendo a la velocidad deseada, no se moverán en forma fluida sino con un parpadeo. Entre más rápida sea la computadora, el desplazamiento va a ser cada vez más fluido. [22.2]

Ambas técnicas tienen sus ventajas y desventajas, y escoger entre uno y el otro depende realmente de lo que se aspire con la aplicación. [22.2]

De manera general, para controlar el tiempo en la aplicación, hay que determinar cuantos *tics* o ciclos de aplicación hay por cada segundo de la máquina. Se obtiene entonces el tiempo actual de la máquina y se almacena en una variable que representa el tiempo de inicio de la aplicación; a partir de este valor, se llevará la referencia del tiempo durante la aplicación. [1.5]

1.3 Motores de Sistemas de Realidad Virtual

En el mundo existen infinidad de entornos para desarrollo de juegos y simuladores que se tomán como referencia para el análisis de funcionalidades que debe tener un *software* de este tipo: estructura de clases, modo de implementación, técnicas y lenguajes de programación utilizados, etc. Pueden citarse como ejemplos el 3D-GameStudio, el CristalSpace, el Fly3D, el Genesis3D, el Nebule, el Ogre, el Shark3D, el Torque y el WorldToolKit. [13]

De este grupo se toman como objeto de análisis el 3D-GameStudio, por ser uno de los más populares, el WorldToolKit, por ser uno de los que adquirió la empresa para desarrollar sus simuladores inicialmente; y el SimpEngine, por ser uno de los más sencillos y que aunque está más orientado a juegos, servirá para en un inicio brindar una idea clara de lo que se persigue con un *engine* de este tipo.

1.3.1 3D-GameStudio. Motor de juegos

3D-GameStudio es un *kit* de desarrollo para la realización de juegos de ordenador.

"Provee de un motor 3D, un motor 2D, un editor de niveles y modelos, compilador de *scripts* y librerías de modelos, texturas, etc. Manipula con igual rendimiento escenas de interior y de exterior. Tiene un motor de iluminación que soporta sombras verdaderas y fuentes de luz en movimiento. El principal objetivo que esta aplicación persigue es que el creador del juego no necesite ser un programador experimentado". [15]

Ofrecen tres posibilidades para crear un juego: juegos diseñados a base únicamente de ratón, para usuarios sin conocimientos de programación; juegos o efectos diseñados con algo de programación utilizando *C-scripts*, para el que

quiere algo más, y juegos o efectos programados en C++ o Delphi, para programadores con experiencia. [15]

Características que implementa [15]

- Modelado: LOD geométrico, Modelado de terreno.
- Visibilidad: Árboles BSP, portales y PVS.
- Iluminación: Fuentes de luz estáticas y dinámicas, sombras estáticas y dinámicas.
- Texturas: *Texture Mapping* y procedural para agua y lava, *Mip-mapping trilineal.*
- Efectos: Partículas, niebla coloreada, transparencia, transformación y deformación suave de mallas.
- Efecto de movimiento lento/rápido, detección de colisiones, inserción de trayectorias para animaciones.
- Soporta cálculo de sombras estáticas y dinámicas.
- Soporte para DirectX. No hay planeado soporte para OpenGL ni para otros sistemas.

1.3.2 SimpEngine. Motor de juegos

SimpEngine es un esqueleto o armazón orientado a objeto y basado en OpenGL, que se puede utilizar para construir juegos. Debido a su simplicidad, no significa que se pueda usar para desarrollar juegos grandes, pero sí para simples juegos rápida y eficientemente. [1.6]

Esencialmente, el *engine* recibe datos a través de un Subsistema de Entrada y envía mensajes a un Subsistema Lógico de Juego, el cual manipula el mensaje y ejecuta el ciclo de juego. En un ciclo simple de juego, el Subsistema responde a las entradas, ejecuta los cálculos físicos necesarios con los objetos del juego,

manipula las colisiones, carga y destruye objetos, mueve la cámara por la escena, y ejecuta cualquier sonido que se necesite durante el juego. [1.6]

Estructura de clases

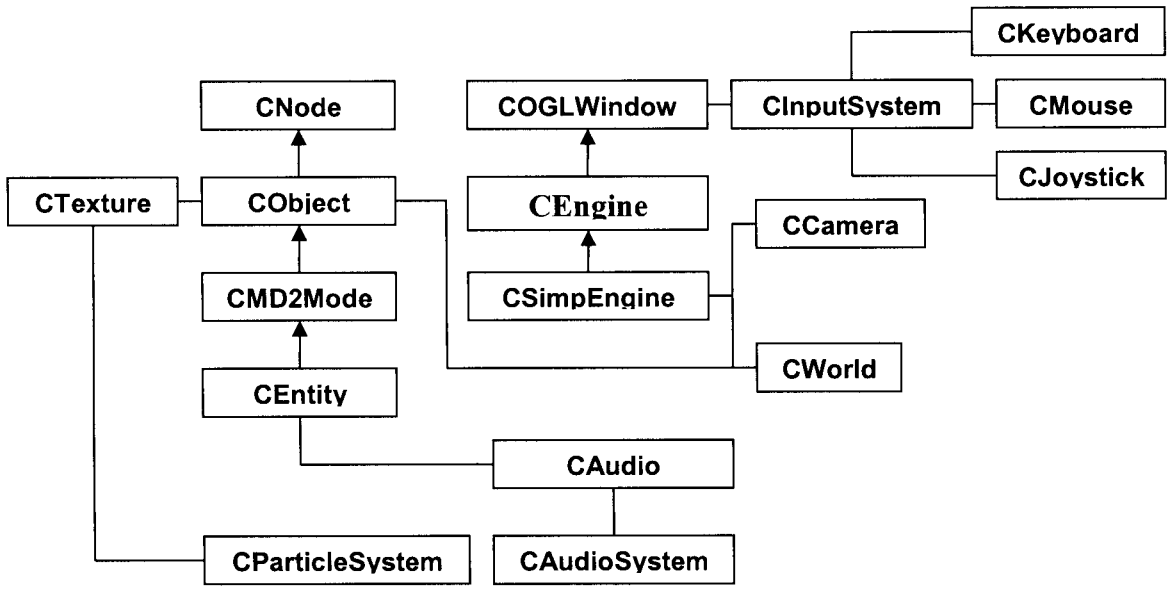

Fig. 4 Estructura de clases del SimpEngine.

Como se nota en la figura anterior, SimpEngine contiene una estructura de clases muy clara, con las funcionalidades y responsabilidades muy bien repartidas. Se tiene por ejemplo que:

COGLWindow manipula mensajes de Windows que son recibidos del sistema operativo, creación de ventanas, algunas entradas básicas, y sirve de padre de CEngine y CSimpEngine. Mantiene el **CInputSystem** que usa **DirectInput** para obtener información del teclado, *mouse*, y estados del *joystick*. El CInputSystem solo es inicializado y apagado por la clase COGLWindow. [1.6]

CNode es un nodo simple de un árbol cíclico que se utiliza para construir la jerarquía de objetos del mundo, para la manipulación de los objetos en el *engine*. La clase CNode brinda la posibilidad de adicionar nodos a cualquiera

existente, eliminar nodos de la jerarquía, etc. Cada nodo padre puede tener solamente un único hijo, y éste será la cabeza de la lista de hijos del padre, como se puede apreciar en la siguiente figura. [1.6]

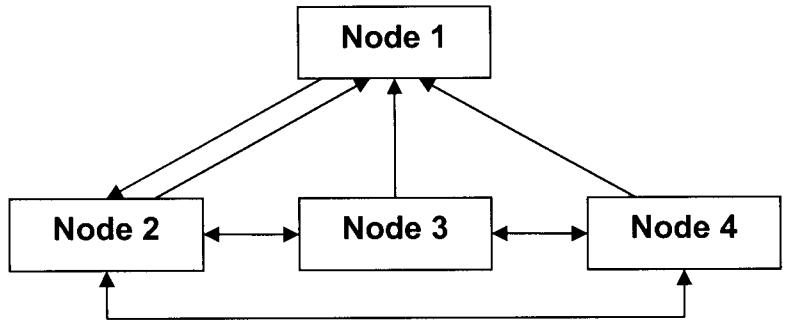

Fig. 5 Ejemplo sencillo de una colección de objetos CNode.

A partir de las clases COGLWindow y CNode se puede construir cualquier extensión que se quiera añadir al *engine*. [1.6]

CObject representa la mínima entidad posible en el SimpEngine. Representa cualquier cosa que se pueda dibujar, mover, colisionar, etc., y como hereda de CNode, se pueden adicionar objetos a cada uno de los otros para crear la jerarquía de objetos. [1.6]

La clase ***CEngine*** suministra el lazo principal de mensajes de Windows, el ciclo del juego, y funciones para manipular cualquier entrada recibida del sistema de entrada. En cada ciclo de juego se chequean los mensajes enviados a la aplicación, se calcula el **tiempo** que ha transcurrido desde el último ciclo de juego, y con éste tiempo se hacen los cálculos físicos y otras funciones basadas en tiempo. [1.6]

La clase ***CEntity*** representa un objeto del juego que es capaz de ser animado por animación por *keyframes*. Brinda un buen soporte para controlar el comportamiento de un modelo MD2, y como hereda de CObject, se puede trabajar con ella de la misma manera. Mantiene la dirección de las entidades

(rotación en Y), el *keyframe* actual, *frame* de inicio y *frame* final, y la velocidad de animación del modelo. [1.6]

La clase ***CParticleSystem*** se utiliza para producir varios efectos de partículas incluyendo explosiones, humo, fuego y precipitaciones. [1.6]

1.3.3 WorldToolKit. Motor de simuladores

WTK es un entorno de desarrollo multiplataforma avanzado, para aplicaciones gráficas 3D en tiempo real con un alto rendimiento, que posee una librería de más de 1000 funciones y herramientas programadas en C para usuarios finales, que les servirán para crear y manejar sus productos. Con una interfaz para programadores de aplicaciones de alto nivel (*Application Programmer's Interface*, API), permite modelar, desarrollar y configurar rápidamente las aplicaciones finales. "WTK aprovecha al máximo las características disponibles de OpenGL". [5.1]

La escena en el WTK se estructura y maneja a través de una arquitectura jerárquica conocida como ***grafo de la escena (Scene Graph Architecture)***. De esta manera, se pueden ensamblar nodos en un árbol jerárquico donde cada uno representa una parte del simulador, indicando como la simulación será interpretada a la hora de dibujar el mundo virtual, lo que le da mucha eficiencia al simulador. [5.1]

Esta estructura permite además el agrupamiento de objetos por tipos o por estado; la alternación de niveles de detalles, (es decir, se pueden especificar las características o detalles para los tres niveles de cercanía a la cámara (cerca, medio, lejos), con mayor facilidad); permite la instanciación de geometrías y de sub-árboles enteros del diagrama de la escena, brindando un mejor uso de la memoria. [5.1]

El ***nodo***, es la estructura mínima que se puede insertar en el grafo de la escena, una clase base de la que heredan todas las demás clases que se manejan en el motor de simulador, esto permite que cualquiera de los objetos pueda ser insertado en dicho grafo y por tanto ser manejado durante la visualización de la escena. [5.1]

El ***lazo de simulación*** del WTK consiste en: leer sensores de entrada, actualizar objetos, y aplicar *render* a una nueva vista de la escena. [5.1]

Clases del WTK ([5.1])

- ***Universe***: es la clase que contiene a los demás objetos, incluyendo a los nodos. Aún cuando se puedan tener varios grafos de escena, solamente existe un Universo.
- ***Geometries***: objetos gráficos visibles en la simulador (cubos, esferas, cilindros, textos 3D…)
- ***Node***: bloques a partir de los cuales se construye la escena.
- ***Polygons***: cada una de las figuras que conforman a las geometrías.
- ***Vertices***: lista de puntos que conforman a los polígonos.
- ***Lights***: luces que se añaden a la escena provocando efectos de iluminación y sombras.
- ***ViewPoints***: puntos de vistas o cámaras, permiten observar la escena desde diferentes ángulos y distancias. Definen la posición y orientación desde la cual las geometrías en la simulación van a ser proyectadas hacia la pantalla.
- ***Windows***: ventanas para mostrar el mundo virtual.
- ***Sensors***: se pueden conectar sensores para transformar nodos, cámaras…
- ***Path***: los objetos *paths* permiten que las geometrías y cámaras puedan seguir "caminos" predefinidos.
- ***Tasks***: se usan para asignar comportamientos a los objetos individualmente. Ejemplo de *tasks* son: Cambiar la apariencia, detectar

intersecciones, provocar otro comportamiento secundario, y adicionar un sensor. Las tareas se eliminan cuando se elimina su objeto.

- ***Motion Links***: conecta una fuente de información de posición y orientación con un objetivo, por ejemplo, un sensor con una cámara. El objetivo se mueve en correspondencia con el estado de la fuente.
- ***Sound***: manipula los sonidos del simulador.
- ***User interface***: elementos para la interacción con el usuario como barras de tareas, menús, cajas de mensajes y otros.
- ***Networking***: la clase *Networking* da la posibilidad de construir aplicaciones distribuidas en red.
- ***Serial Port***: maneja un grupo de funciones que simplifican las tareas de comunicación con los puertos en serie.

Tipos de nodos

La complejidad y gran eficiencia del manejo del mundo virtual del WTK radica en los tipos de nodos con que se construye el grafo de la escena.

La escena del WTK se concibe como geometrías, luces, *fog* (niebla, contiene todos los demás efectos ambientales), e información posicional necesaria para representar los objetos en la escena, [6.1]. Estos componentes se representan a través de nodos de diversos tipos que se pueden agrupar en dos grandes tipos: nodos de contenido y nodos de agrupamiento.

Los ***content nodes***, o nodos de contenido, representan a las geometrías, luces y nieblas, es decir, aquellos que contienen datos y que forman, de alguna manera, entidades visibles de la escena. [6.1]

Los ***grouping nodes***, o nodos de agrupación, representan información que modifica a los nodos de contenido. Son los más importantes: el ***group node***, que sirve para agrupar nodos por cualquier criterio; el ***separator node***, que se utiliza para encapsular o separar nodos o grupos de nodos, y de esta manera

evitar la propagación de estados de una rama a otra; el ***transform separator node***, que solamente encapsula el estado de transformación, y no el estado de luz o niebla; y el ***procedural node***, que incluye al ***switch node***, que conmuta las ramas, es decir, indica por cuál rama se va a recorrer el grafo en determinado momento; y al ***LOD (level-of-detail) node***, a partir del cual generalmente se abren tres ramas con distintas características para el nivel cercano, medio o lejano, y en dependencia de la distancia a la cámara, se recorre solamente una de estas ramas. [6.1]

El WTK trabaja las **colisiones** con *bounding boxes*. [6.1]

1.3.4 Ventajas y desventajas de los Motores de RV

De las múltiples características que presentan cada uno de los motores tomados como referencia, se analizarán las relativas a los objetivos que se plantearon como solución a los problemas que dio origen a este proyecto. Nótese que de las funciones generales que realizan estos *engines* durante un ciclo de juego (recoger las entradas, mover jugadores, ejecutar la inteligencia artificial, hacer cálculos físicos, ejecutar sonidos y aplicar *render* de la escena) [1.6], la parte que abarca este proyecto es la relativa al movimiento y animación de objetos, y al *rendering* de la escena, es decir, la visualización del mundo virtual.

En cuanto al soporte de modificadores de la escena, los cuales cambian cualquier aspecto visual quedando sensación de animación, en el 3D-GameStudio se le da al programador la posibilidad de insertar *scripts* que cambien los atributos de los objetos, tanto de posición y orientación como visuales (colores, texturas...), con lo que se pueden lograr animaciones simples o rígidas. Además se pueden insertar *path* o caminos y ser asociados a objetos para que sigan dicho movimiento.

Para el WorldToolKit existe también el mecanismo de asignación de "*task*" o tareas a los objetos, estas se asocian a funciones que pueden provocar cambios de apariencia o darle movimiento; también se asignan *path* a los objetos.

Los tres motores soportan la deformación de mallas (en el caso del 3D-GameStudio también por huesos para la versión del 2004), aunque no tienen atención especial con los personajes en cuanto al soporte de las cualidades físicas y emocionales, que tanto realismo le dan a los juegos y simuladores añadiéndole variaciones a las acciones de los personajes, considerando por ejemplo, su estado anímico. [19]

Sobre la utilización transparente de más de una librería gráfica, ninguno cumple con este requerimiento: WTK solamente trabaja con la OpenGL al igual que SimpEngine, mientras que el 3D-GameStudio trabaja con DirectX, por lo que ninguno de ellos cubre el objetivo de soportar ambas librerías.

Sobre la visualización eficiente de la escena y el manejo del grafo de la escena, en el 3D-GameStudio se trata la visibilidad a través de árboles BSP, portales y PVS (espacios potencialmente visibles), con los que se logra un rechazo importante de partes de la escena no visibles, lo que alivia enormemente al procesador gráfico.

En el SimpEngine existe un único tipo de nodo con el que no se logra identificar los recorridos por el árbol de la escena, esto provoca que en cada ciclo se recorra el árbol completo restándole eficiencia al simulador cuando se trata de una aplicación muy grande, donde invierte mucho tiempo y recursos visualizar toda la escena.

El WTK, en cambio, tiene una variedad de nodos que le permite un trabajo eficiente con la representación de la escena, pues especifican muy bien a qué partes del grafo de la escena acceder según la visibilidad, optimizándose el

recorrido por el grafo, lo cual resulta importantísimo cuando la aplicación toma dimensiones considerables.

Los tres motores analizados son poco flexibles, no le permiten al programador agregar módulos propios por lo que se necesita depender de nuevas versiones o actualizaciones.

1.4 Lenguajes de programación

En esta sección del capítulo se analizarán dos de las bibliotecas gráficas que compiten en el mundo del desarrollo de gráficos 3D, y el lenguaje de programación a utilizar.

1.4.1 Librerías gráficas. OpenGL y DirectX

A continuación se presentarán las caracteristicas fundamentales de cada biblioteca:

OpenGL

OpenGL es una librería gráfica que provee a los programadores de una interfaz de acceso al *hardware* (HW) gráfico. Es poderoso, con *rendering* a bajo nivel y una librería de *software* de modelamiento, disponible en la mayoría de las plataformas, con un amplio soporte de HW. Es diseñado para ser usado en cualquier aplicación gráfica, desde juegos y simuladores hasta modelaciones CAD (*Computer Aided Design*). [1.1]

Arquitectura OpenGL

OpenGL es una colección de cientos de funciones que dan acceso a todas las ventajas ofrecidas por el HW gráfico. En el corazón del OpenGL está el ***rendering pipeline***, (tubería de *rendering:* serie de pasos que se siguen para aplicar el *render*, y que son manipulados por OpenGL), como se muestra en la siguiente figura. [1.1]

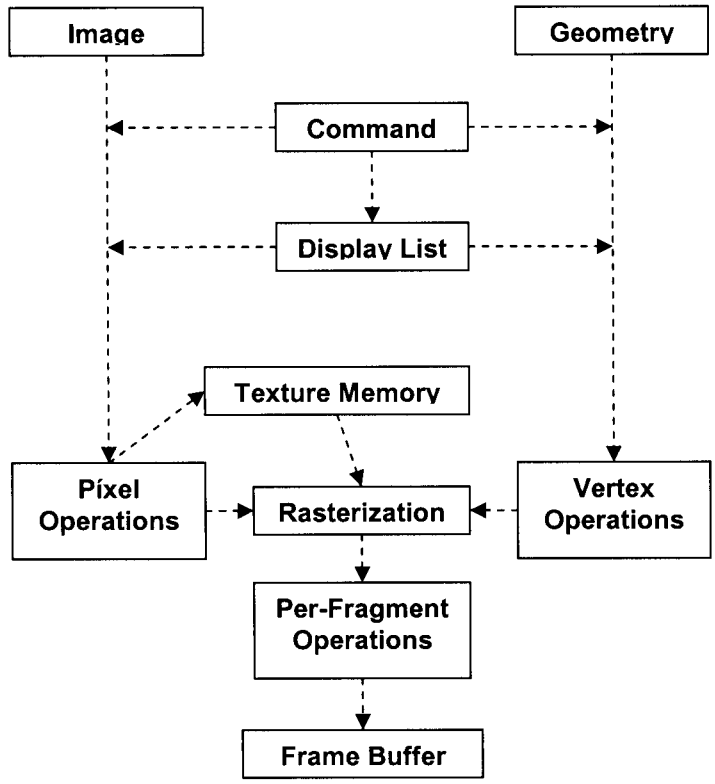

Fig. 6 Rendering Pipeline. Tubería de rendering.

Bajo Windows, OpenGL brinda una alternativa consistente en usar *Grafics Device Interface* (**GDI**, Interfaces de Dispositos Gráficos), diseñada para hacer el HW gráfico completamente invisible al programador a través de capas de abstracción (esto aminora la velocidad). Las GDI se pueden ignorar y lidiar directamente con el HW gráfico (ver figura 7). [1.1]

Como OpenGL es una API gráfica, no soporta directamente ningún tipo de ventanas o menús, por lo que se necesitan manipular estos objetos. Cada sistema operativo usualmente tiene un grupo de extensiones que permiten este trabajo. En sistemas Unix, esto se logra con **GLX**. En Microsoft Windows, sin embargo, se usa un grupo de funciones llamadas ***wiggle functions*** (ver figura 7). Cada una de estas funciones tiene prefijo **WGL** por *wiggle*. [1.2]

Windows define además un conjunto de funciones API Win32 que son usadas como interfaz con OpenGL. [1.2]

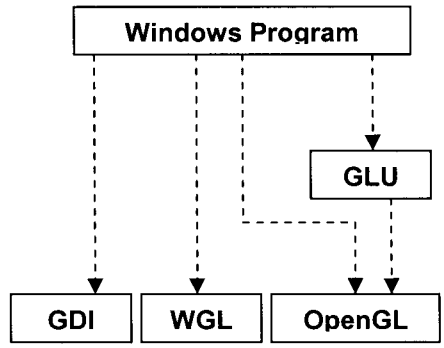

Fig. 7 Jerarquía OpenGL API bajo Windows.

OpenGL, no obstante, ofrece su propio grupo de librerías conocido como *OpenGL Utility ToolKit* (**GLUT**, Herramientas y Utilidades de OpenGL), con soportes disponibles en la mayoría de las plataformas y funcionalidades básicas en el área del trabajo con ventanas. GLUT mantiene la independencia de las plataformas, es decir, se puede mover fácilmente una aplicación basada en GLUT de Windows hacia Unix, con unos pocos cambios (ver figura 7). [1.2]

DirectX

"DirectX es un intento de Microsoft de brindar un acceso "directo" al HW en el entorno del sistema operativo Windows, a través de un conjunto de APIs que controlan un grupo de funciones que acceden al HW o lo simulan si no existe." [1.1]

Dichas funciones incluyen soporte para aceleración gráfica 2D y 3D, control de dispositivos de entrada, funciones para mezclar y probar sonidos y música, control para juegos en red y *multiplayers*, y control sobre varios formatos de *streaming* de multimedia (*streaming* es un método de transferencia de datos continuamente, que permite mostrar los datos antes de que el fichero entero haya sido transmitido). [1.1]

Los componentes APIs que manipulan estas funciones son: DirectDaw, Direct3D, DirectInput, DirectSound, DirectMusic, DirectPlay, DirectShow. [1.1]

La filosofía de Microsoft es hacer una multimedia rápida, independiente de los dispositivos y rica en funciones para el sistema operativo Windows. [1.1]

Arquitectura DirectX

DirectX usa dos *drivers* o manejadores: la capa de abstracción de HW (HW *abstraction layer*, **HAL**) y la capa de simulador de HW (HW *emulation layer*, **HEL**; HEL ignora el HW e implementa sus funcionalidades). Cuando DirectX es inicializado, chequea el HW para ver si tiene ciertas potencialidades, y en dependencia de ello usa el HAL o el HEL. Esta arquitectura permite ser expandida fácilmente en un futuro HW (ver figura 8). [1.1]

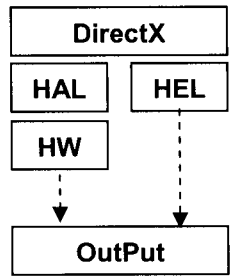

Fig. 8 Arquitectura HAL/HEL de DirectX.

OpenGL contra DirectX

La primera diferencia entre estas librerías gráficas es que DirectX posee más que solamente componentes gráficos, pues incluye herramientas para sonidos, entradas, música, redes, y multimedia. OpenGL, por otra parte, es estrictamente una API gráfica.

Ambas APIs usan la tradicional *grafics pipeline* (tubería gráfica). Es el mismo *pipeline* que se diseñó desde las primeras computadoras graficas, que se ha ido

modificando de acuerdo a los avances de HW aunque sin cambiar la idea básica. [1.1]

"Ambas APIs describen los vértices como un grupo de datos consistente en coordenadas en el espacio que definen la localización del vértice. Las primitivas graficas (puntos, líneas, y triángulos) están definidos como un grupo ordenado de vértices. Sin embargo, la diferencia entre las APIs está en cómo los vértices son combinados para formar las primitivas: cada uno lo maneja diferente". [1.1]

La siguiente tabla ofrece las características distintivas de ambas librerías ([1.1]):

Tabla 1 Características distintivas OpenGL vs. DirectX

Características	OpenGL	DirectX
Múltiples sistemas operativos	Sí	No
Mecanismo de extensión	Sí	Sí
Desarrollo	Múltiples Compañías	Microsoft
Two-Sided Lighting	Sí	No
Textura de volumen	Sí	No
Z-Buffers independiente del *hardware*	Sí	No
Buffers de acumulación	Sí	No
Antialiasing de pantalla completa	Sí	Sí
Difuminación de movimiento	Sí	Sí
Profundidad de área	Sí	Sí
Rendering estéreo	Sí	No
Tamaño de punto y ancho de línea	Sí	No

Picking	Sí	No, pero con funciones de utilidad
Curvas y superficies paramétricas	Sí	No
Caché de geometrías	Listas de visualización	*Buffers* de vértices
Simulación de *Software*	Si el HW no está presente	Permite que la aplicación determine
Interfaz	Llamadas a procedimientos	COM (*Component Object Model*)
Actualizaciones	Anuales	Anuales
Disponibilidad de código fuente	Implementación de muestra	Punto de inicio en Microsoft DDK (*Driver Development Kits*)

1.4.2 Lenguaje de desarrollo C++

Existen principalmente tres lenguajes que se utilizan para desarrollar aplicaciones gráficas profesionales en 3D: Lenguaje Ensamblador, C y C++, por ser los que con más velocidad ejecutan el código (menor costo de ejecución del programa). A éstos se ha unido recientemente el Java como una opción para el desarrollo de este tipo de aplicaciones.

Es decisión de la entidad cliente, implementar este proyecto mediante el lenguaje C++, que ha estado en su línea de trabajo con magníficos resultados.

Es el lenguaje en que se tiene mayor experiencia por parte del equipo de SIMPRO-UCI, futuros desarrolladores del sistema que ocupa este proyecto.

Si se estudian las características de este lenguaje, se puede apreciar lo acertado de la elección, dado que C++ es un lenguaje de programación de propósito

general, especialmente indicado para la programación de sistemas por su flexiblidad y su potencia. Es uno de los más utilizados en las comunidades de desarrolladores de *software*, incluyendo la programación gráfica. [22.1]

C++ es la evolución de C adaptada a la programación orientada a objetos. Tiene algunas cuestiones más pulidas como un control más estricto en el manejo de tipos de datos, y otras características que ayudan a la programación libre de errores. [22.1]

En general puede llegar a ser un lenguaje tan rápido como C (el más rápido después del Lenguaje Ensamblador), sin embargo, si se maneja herencia múltiple, funciones virtuales y polimorfismo en forma inadecuada, o se accede mucho en niveles de profundidad en la llamada a objetos (Objeto1.Objeto2.Objeto3.Objeto4...), puede llegar a hacerse un poco más lento, lo cual no es conveniente para una aplicación en tiempo real. [22.1]

Conclusiones

En el transcurso de este capítulo, como base para el entendimiento del tema en que se desenvolverá este proyeto, se formuló un concepto de Sistema de Realidad Virtual a partir de caracterizaciones de diferentes autores. Además se mostraron las principales características de estos sistemas y las técnicas, tecnologías y tendencias actuales más utilizadas para su desarrollo.

Se logró también entender el funcionamiento básico de los motores de simuladores a través del estudio de tres casos, y se definieron las principales deficiencias que deben ser superadas por el futuro producto a obtener con el presente trabajo.

2

Capítulo 2 Soluciones técnicas

Introducción

En el presente capítulo se proponen soluciones técnicas para el funcionamiento del motor de simulación, y soluciones específicas para lograr la visualización eficiente 3D en tiempo real y la incorporación de personajes animados.

2.1 Manejo de los objetos de la escena

El manejo de los objetos de la escena se tratará a través de un grafo de escena, el cual inicialmente se asemejará a un árbol general, pero podrá ser más complejo a conveniencia del programador, en caso de que desee crear nuevos nodos que propicien una navegación típica de grafos.

Dicho grafo estará conformado por un grupo de nodos que heredarán de un padre común (*Node*), y las clases externas al grafo harán llamadas a un *Node* transparentemente. Ésto permitirá que cualquier nodo (o clase en general) que herede de *Node*, se pueda insertar en el grafo, y de esta manera, se le da la posibilidad al usuario de la biblioteca de crear nuevos tipos de nodos y configurar el grafo a conveniencia. Por ejemplo, si se quisiera que un determinado grupo de bjetos se almacenen en un árbol binario de partición espacial (BSP Tree), pueden crearse *BSPTree-nodes* que se puedan adjuntar a alguna parte específica del grafo de la escena, y para esta parte, la navegación sería de una manera diferente e independiente del resto del árbol.

No obstante se le brindará al programador un grupo de nodos con los cuales se pueden representar los principales objetos de la escena y sus características. Estos nodos serán esencialmente nodos agrupadores y nodos de contenido.

Tipos de nodos

Los nodos de contenido estarán especializados (existirán "nodos geometría", "nodos cámara", "nodos luz", "nodos personaje" y "nodos hueso") y serán los encargados de darle el tratamiento adecuado a su contenido. Dichos nodos de admitirán transformaciones locales independientes de los demás nodos pertenecientes a su grupo. Por ejemplo, el nodo hueso coordinará el proceso de deformación de la malla.

En el caso de las geometrías, es común que en un escenario varios cuerpos tengan el mismo modelo, y es ineficiente almacenar una copia del modelo por cada objeto de la escena; para solucionar esta situación, los "nodos geometría" contendrán una referencia a un modelo ubicado en una colección, y cada uno calculará por separado las transformaciones que se le aplicarán a la geometría durante el *rendering*.

Los nodos grupo se encargarán de reunir los nodos y contener características comunes a todos, como se explicará más adelante.

La siguiente figura muestra un grafo simple, donde existe una cámara, un auto y dos árboles. Los árboles, en caso de ser "del mismo modelo", como se dijo antes, referenciarán a una misma geometría. En el grupo auto, cada rueda tiene información independiente, por tanto se les podrá aplicar rotaciones (como en la vida real), y una traslación al grupo auto moverá todas sus partes. Véase además la posibilidad de añadir otros objetos a un grupo, por ejemplo, los objetos luces en el grupo auto, si se ubican convenientemente "en el frente" del auto, y no se les da movimiento aparte, funcionarán como sus luces.

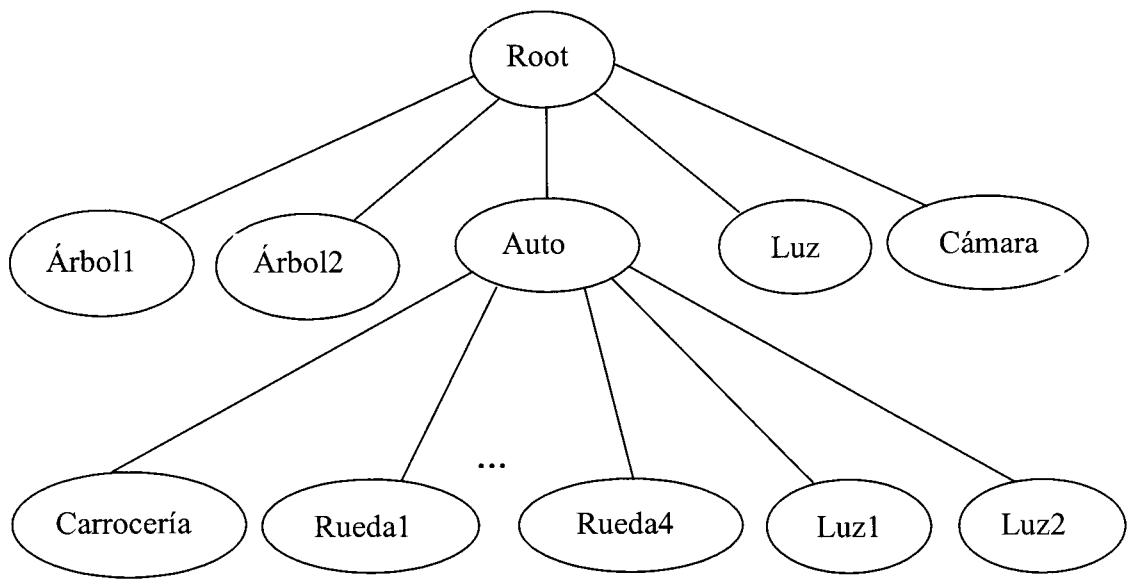

Fig. 9 Ejemplo de un grafo de la escena simple.

2.1.1 Actualización de la escena

Para la actualización de los objetos de la escena, cada nodo almacenará la información necesaria para actualizar su contenido (ya sea un objeto o un grupo de nodos).

La información espacial (posición, orientación y escala) de un nodo, se almacenará en matrices de transformación. Aunque se puede almacenar la matriz homogénea como un arreglo de 4x4 (siguiendo el formato gráfico de las matrices de transformación como se explicó en el epígrafe de Transformaciones geométricas), se ha demostrado en la práctica que ésto no es recomendado pues siempre la última fila de la matriz es [0 0 0 1], lo que consume más memoria, y lleva más iteraciones cuando se realizan operaciones con las matrices [8.2]. En solución a esto, se almacenará la matriz homogénea como un arreglo de 3x3, correspondiente al bloque superior izquierdo de la matriz de 4x4 (en esta parte se contiene la rotación y la escala [8.2]), y en un vector 3x1 se almacenará el componente traslación de la matriz. Esta descomposición de la matriz se puede ver en la figura siguiente:

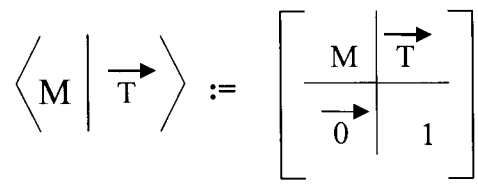

Fig. 10 Descomposición de la matriz homogénena.

A esta información espacial se le llamará ***Estado Geométrico (EG)*** del nodo, y cada nodo tendrá dos estados geométricos, el ***EG local*** (relativo a su padre) el ***EG mundial*** (respecto a toda la escena).

En el ejemplo del auto, el EG de las ruedas respecto al "grupo auto" es su EG local; al combinar el EG mundial de su padre ("grupo auto") con el EG local propio, se obtiene el EG mundial de la rueda, que es el estado con el que realmente se representa el objeto en la escena. A su vez, el EG mundial se

grupo auto, será el producto de la combinación de su EG local con el EG mundial de su nodo padre, y así recursivamente. Es decir, se representan los objetos según el resultado total de las transformaciones desde la raíz del grafo hasta cada objeto. El siguiente pseudocódigo demuestra el proceso de actualización del EG de un nodo:

```
ActualizarEG ():
    1- EGLocal = ActualizarEstadoGeometricoLocal();
    2- EGMundialPadre = ObtenerEGMundialPadre();
    3- EGMundial = EGLocal * EGMundialPadre;
Fin
```

La resto de la información con que se representarán los objetos (texturas, estados de *alpha*, modo de visualización (malla, sólido), material...), llamada ***Estados de Render (ER)***, también se almacenará en cada nodo, y el mecanismo de transmisión de la información será parecido: los ER de los objetos en los nodos hojas dependerán de los ER de los nodos recorridos desde la raíz del grafo hasta ellos.

```
ActualizarER ():
    4- ERLocal = ActualizarEstadoDeRenderLocal();
    5- ERMundialPadre = ObtenerERMundialPadre();
    6- ERMundial = ERLocal * ERMundialPadre;
Fin
```

Al llegar a las hojas se tienen acumulados los estados de *render* a aplicar. El proceso de cambio de estados de *render* a menudo cuesta caro, por ejemplo, cuando se le cambia la textura a un cuerpo, si ésta está en la memoria del sistema pero no en la de video, pasar la información de una a otra es un proceso lento, por lo que es conveniente que se seleccionen determinado estados de *render* a los cuales se les desea minimizar los cambios, y el renderer decidirá de acuerdo a esto, si cambiar o no su información interna.

La tercera información que almacenarán los nodos será el ***volumen frontera (VF)*** de su objeto, la cual, como ya se ha explicado, se utilizará para determinar con mayor rapidez si un objeto se encuentra en el cono de visión de la cámara.

La actualización del VF de un nodo de contenido, se hará modificando su VF base según el EG mundial del nodo:

```
NodoContenido.ActualizarVF ()
    1- CopiaVFBase = ObtenerVFBase();
    2- VF = CopiaVFBase.Modificar (EGMundial);
Fin
```

Y la de un nodo grupo, se hará combinando los volúmenes fronteras de sus nodos hijos:

```
NodoGrupo.ActualizarVF ()
    1- VF = 0;
    2- Para cada Nodo Hijo i:
        2.1- VFH = ObtenerVFHijo (i);
        2.2- VF = VF + VFH;
Fin
```

Por supuesto, para actualizar el VF de un nodo cualquiera, primero debe haber sido actualizado su EG; y para actualizar el VF de un nodo grupo, deben haberse actualizado los VF de sus nodos hjos.

Como no todos los objetos tienen en realidad volumen frontera ni estados de *render* (ejemplo, las luces), existe un "nodo dibujable" del cual heredan los nodos que contienen este tipo de información.

Para el control de los estados de un nodo (EG y ER), se utilizarán controladores cuya función es determinar el estado local de cada nodo en un instante dado de la simulación. Existirán controladores de animaciones y de caminos, que

determinarán la información espacial de un nodo; y controladores de tareas, que determinarán, según las tareas asociadas al nodo, alguna transformación a aplicar, que puede ser una transformación espacial, o una alteración de los estados de render. Un nodo puede tener varios controladores, o ninguno.

Transmisión de la información

La actualización de estados de todo el árbol será un proceso recursivo: se bajará por el árbol actualizando el EG mundial de cada nodo (para ésto, cada nodo determinará su EG local y lo multiplicará por el EG mundial de su padre) y su ER; y al llegar a las hojas, se regresará a la raíz actualizando los volúmenes fronteras. El proceso sería como sigue:

ActualizarNodo (NodoActual):

1- NodoActual.ActualizarEG ();
2- NodoActual.ActualizarER ();
3- Para cada Nodo Hijo i
   3.1- ActualizarNodo (Nodo Hijo i);
3- NodoActual.ActualizarVF();

Fin

La actualización del grafo completo se haría a través de la llamada: ActualizarNodo (Nodoraíz).

Los nodos agrupadores están concebidos no sólo para agrupar otros nodos, sino también para dirigir la navegación por al grafo. Contienen variables de control que optimizarán el recorrido por la rama que éste encabeza, por ejemplo, se mantendrá una bandera que indique si el nodo grupo está actuando como *switch node* o nodo conmutador, en cuyo caso se determinará cual es, de las ramas que parten de él, la activa en ese momento, recorriéndose solamente dicho subárbol. Esto podría utilizarse, por ejemplo, para recorrer un edificio, donde

existe un nodo grupo" Edificio", y nodos hijos "Pisos", y en el nodo "Edificio" se indicaría por cuales de los pisos se debe recorrer el árbol en ese momento.

2.1.2 Representación eficiente de la escena

Una vez actualizados los estados de cada nodo de la escena, el proceso de representación o dibujado es sumamente sencillo, basta con determinar si el nodo está visible (detección de colisiones de los volúmenes con el *frustum* de la cámara), y en este caso, se mandaría a dibujar; en caso contrario, se rechaza la rama correspondiente al nodo que no queda "en cámara", disminuyéndose así la mayor parte del recorrido por el grafo.

```
RepresentarNodo (NodoActual, CámaraActiva, Renderer):
    1- Si NodoActual.VF colisiona con CámaraActiva.Frustum
        1.1-  Si NodoActual.Tipo = NodoGrupo
              Para cada Nodo Hijo i
              1.1.1- RepresentarNodo (Nodo Hijo i, CámaraActiva, Renderer);
        1.2-  Sino
              1.2.1- Renderer.Dibujar (NodoActual);
    2- Sino, salir.
Fin
```

2.2 Animaciones

Las animaciones por deformación de mallas se realizarán con la técnica de animación por esqueleto, que es la que mejor logra la diversidad y fluidez del movimiento que requieren los productos de calidad comercial. Actualmente, esta novedosa técnica está siendo insertada en los más prestigiosos sistemas con posibilidades de animaciones de deformación de mallas, como por ejemplo el 3D-GameStudio, el cual la adicionó en la nueva versión sacada este año.

Para la deformación de una malla por huesos, primeramente se recorre recursivamente el esqueleto, actualizando las matrices de transformación de cada hueso. Este recorrido se hace durante la actualización del grafo de la escena, ya que los huesos están representados como nodos hijos de un "nodo personaje", y bastaría con hacer una llamada al método "ActualizarNodo(NodoPersonaje)", visto en el epígrafe "Actualización de la escena".

La actualización de cada vértice influenciado por un hueso, se hace multiplicando el vértice sin modificar, por la matriz de transformación del hueso ya calculada, por el peso que ejerce el hueso sobre el vértice.

Bone.ActualizarVertices():

1- Para cada Vértice_i

1.1 - NuevoVertice_i = Vertice_i * Matriz * PesoVertice_i

1.2 – Malla.Vertice_i += NuevoVertice_i

Fin

El proceso de actualización de todos los vértices de la malla consiste entonces en que cada hueso modifique los vértices que le pertenecen:

ActualizarMalla():

1- Para cada Vertice_i

1.1- Vertice_i = 0

2- Para cada Hueso_i

2.1- Hueso_i.ActualizarVertices()

Fin

La velocidad de las animaciones para cada personaje, se verá afectada por las características individuales de cada uno en la escena; estas son: su estado físico y emocional. Para ello, se dará por cada animación un rango de velocidades permisibles y se hallará la que corresponda a cada instante, según el promedio de los valores que presenten el estado físico y el temperamento del personaje, en escala de 0 a 1.

El estado físico de un personaje se modifica de manera que 0 describe el estado inanimado por agotamiento límite o muerte de un personaje, y 1 el tope de las posibilidades para realizar una animación a velocidad máxima.

El temperamento de un personaje, a su vez, se expresa a través de diferentes tipos de temperamentos (para esta versión serán Sanguíneo, más pausado, y Colérico, explosivo) que respondan a valores dentro de la escala.

La selección y cálculo del *frame* correspondiente en dependencia de estos parámetros y del tiempo acumulado de la aplicación, se llevará a cabo por controladores de animaciones de *skin* que estarán asociados a objetos cuyos modelos soporten este tipo de animaciones, es decir, que sean modelados por huesos.

Otro tipo de animación que se soportará será la traslación por *paths* o caminos que se podrán asociar a objetos a través de controladores de *paths*. Dichos controladores se encargarán de calcular la nueva posición y orientación de los objetos. Los *paths* se le podrán aplicar tanto a objetos geométricos como a cámaras y luces.

Se le llama *path* a la lista de vectores que definen una trayectoria a través de la escena, y que una vez asignado a un objeto, el sistema se encargará de desplazarlo siguiendo el recorrido indicado por los vectores.

Otra manera de dar soporte a los cambios de estados geométricos de los objetos de la escena (posición, orientación y escala) y en un futuro, a cambios de estados de *render* (textura, *alpha*...), será a través de tareas.

Las tareas permitirán ejecutar un código escrito por el programador, que se interpretará como una acción determinada; dicha acción será calculada por controladores de tareas que estarán asociados a los objetos a los que se les quieran aplicar la transformación.

En los pseudocódigos del epígrafe "Actualización de la escena", en la actualización de estados se hacían las siguientes llamadas:

EGLocal = ActualizarEstadoGeometricoLocal() en ActualizarEG()
ERLocal = ActualizarEstadoDeRenderLocal() en ActualizarER()

A estos métodos se les debe pasar el tiempo de la aplicación, necesario para determinar el siguiente "paso" o *frame* en las animaciones. Es en estos métodos en los que el nodo llama a sus controladores, y estos buscan, según el tiempo indicado, el *frame* correspondiente en las animaciones, *paths* y/o tareas asociadas. "ActualizarEstadoGeometricoLocal" llama a los controladores de animaciones y *paths*, y "ActualizarEstadoDeRenderLocal", a los controladores de estados de render.

Como se dijo en el epígrafe "Manejo de los objetos de la escena", las instancias de geometrías en la escena pueden responder a un mismo modelo, por lo que los nodos de este tipo referencian a mallas comunes.

De la misma manera, varios personajes pueden estar ejecutando la misma animación (aunque a velocidades diferentes, y por tanto, mostrando *frames*

diferentes), y varios objetos pueden estar ejecutando el mismo *path* o las mismas tareas. La solución a esto es semejante: los controladores referencian a animaciones, *paths* o tareas que están en una colección, y cada uno toma de ellas la información necesaria para el curso de su animación, lográndose la diversidad de movimientos deseada.

Para esto, es necesario que cada frame parta de la posición inicial del modelo, es decir, que cada nuevo movimiento no sea a partir del movimiento anterior (habría que almacenar la malla con la última transformación), sino de la posición inicial, y de esta manera poder transformar la malla inicial común para todos estos objetos. Durante el proceso de carga y construcción de las animaciones se debe verificar que cumplan con este requisito.

2.3 Consideraciones técnicas generales

El sistema, en respuesta a estas necesidades y a las exigencias de velocidad y realismo de simuladores, se basará en el *rendering* en tiempo real.

Permitirá **modelado** por superficies poligonales (preferiblemente triángulos), y soportará los modelos procedurales (sistemas de partículas). Se desarrollará un algoritmo de eliminación de polígonos por niveles de detalles (LOD), y se hará *rendering* en modo alámbrico y de superficie.

Se soportará el modelado de terrenos como una malla de polígonos con alturas, las cuales se almacenarán en *Heightmaps*.

El sistema soportará la alternación entre ambos tipos de **proyecciones** (paralela u ortogonal y perspectiva) a través de las propiedades de las cámaras de la escena.

Las **transformaciones,** como se trató en el epígrafe "Actualización de la escena", se realizarán a través de matrices 3x3 y vectores, para las operaciones necesarias durante la actualización de los estados geométricos. En el momento de ser enviados al *render* los cuerpos, se convertirán estas matrices a las típicas matrices 4x4 con las que trabajan los renderer más conocidos, y que piensa soportar este sistema. No se realizarán transformaciones no lineales.

El sistema soportará simultáneamente los dos volúmenes más sencillos de **colisiones**, *Bounding Spheres* y *Bounding Boxes,* de manera que se puedan escoger cualquiera de las dos, o ambas. El resto de las técnicas y tipos de fronteras se consideran demasiado complejas (por el tiempo de cálculo que pudieran requerir según la irregularidad de la geometría a tratar) e innecesarias dado el entorno de un simulador, donde el usuario generalmente, dado el dinamismo, no está al tanto de la exactitud de las colisiones. En una primera versión del producto se utilizarán los volúmenes de colisiones solamente para

determinar intersecciones con el volumen de visión de la cámara, y no para determinar colisiones entre los cuerpos, por tanto no se tendrán en cuenta las respuestas a las colisiones.

Las entidades geométricas del sistema tendrán las coordenadas (u,v) para dar soporte a la aplicación de **texturas** a través de mapas de texturas.

Para la **iluminación** se emplearán los cuatro tipos básicos de luces (ambiente, directa, punto y *spot*) con tres tipos de colores (ambiente, difusión y especular).

Se soportará el módulo de **efectos ambientales** en sus dos técnicas más conocidas: sistema de partículas (con la creación de los objetos que pueden ser utilizados como partículas (puntos, líneas, polígonos...)); y animaciones predefinidas (puede ponerse un plano al que se le cambie la textura controlado por una tarea).

El **tiempo** se controlará a través de un *timer* que obtendrá en cada ciclo el tiempo actual de la máquina, el cual se tomará como referencia del tiempo que ha durado la aplicación. Según la cantidad de *frames* por segundo con que se desee mostrar el simulador, se le asignará un tiempo de dibujado por *frame*, así, cada vez que se realice un ciclo de dibujo, se esperará el tiempo asignado y se repetirá la operación. Esto responde a la técnica de sincronización en base al *framerate*, aunque la aplicación estará preparada para hacer los cálculos para cualquier tiempo dado, de manera que, aunque no se fije el *framerate*, se podrá hacer la sincronización en base al tiempo (si no tiempo por frame, simplemente se repetirán los ciclos sin hacer espera). Es decir, se podrá trabajar de ambas maneras, y se dejará que el programador fije o no el *framerate*.

La aplicación se preparará para ambas **librerías gráficas**, OpenGL y DirectX, aunque se dará flexibilidad a que se puedan añadir en un futuro otras librerías gráficas. Excepto el proceso de *rendering*, que se realizará con la librería gráfica

que se esté usando, el resto del trabajo se hará a través de funciones programadas puramente en el **lenguaje de desarrollo** C++.

El diseño e implementación se corresponderá con la filosofía de Programación Orientada a Objetos.

Conclusiones

Finalmente, se quiere destacar lo novedoso de algunas desiciones tomadas en este proyecto, como son: la posibilidad de trabajar con más de un renderer; el que las instancias de objetos de la escena referencien a los modelos y a las animaciones en lugar de hacer copias de ellos; la inclusión del temperamento de los personajes en su animación; y la posibilidad de trabajar el tiempo en base o no al *framerate*.

Con este capítulo quedan sentadas las bases técnicas por las que se regirá el sistema. De éstas, se desprenden datos que deben contener los objetos para la realización de los algoritmos, y por tanto, partiendo de ellos, se diseñará una estructura de clases en correspondencia con las técnicas citadas y para lograr los objetivos del proyecto.

3

Capítulo 3 Descripción de la Solución Propuesta

Introducción

En este capítulo se comienza a tener la visión del sistema a desarrollar. Aquí se inicia la concepción práctica del producto a elaborar, sobre la base de las dificultades, necesidades y caracteristicas organizacionales del cliente, conociendo ya las técnicas a utilizar descritas en el capítulo anterior.

3.1 Objeto de estudio

Es **objeto de estudio** de este proyecto, el proceso de producción de simuladores llevado a cabo en la empresa SIMPRO, como resultado de las actividades de cada área de la empresa (véase anexo "Organigrama de la Empresa") y la interacción entre ellas.

En el Departamento o Dirección de Investigaciones Básicas (aseguramiento matemático básico), se investigan las tecnologías y algoritmos eficientes que serán utilizadas en los SRV.

En la Dirección de Software se diseñan y producen las herramientas y sistemas que serán empleados para la visualización de los escenarios virtuales de los simuladores, aplicando los resultados de la Dirección de Investigaciones Básicas.

En las áreas de Mecánica, se investigan, diseñan y producen los equipos en que se empotrará el sistema informático del simulador.

En el área de Electrónica se trabaja todo lo relativo a los componentes electrónicos de los simuladores, el audio, circuitos integrados, tarjetas apropiadas, etc., que permitirán la interacción entre la parte mecánica y la parte informática.

SIMPRO, después de haber comercializado inicialmente sus productos con éxito, ha decidido por parte de la Dirección de *Software*, trazar las siguientes estrategias para proyectarse en el futuro inmediato:

- Exportar simuladores profesionales que utilicen la Realidad Virtual.
- Convertirse nacionalmente en líder de Investigación y desarrollo de simuladores que utilicen la Realidad Virtual.

La visualización del mundo virtual, que es el **campo de acción** de este proyecto, se hace actualmente con una herramienta implementada por la empresa, la cual obstaculiza el cumplimiento de las estrategias dado que:

- Utiliza técnicas de visualización ineficientes.
- No incluye la animación de personajes.
- Está dirigida a una única biblioteca gráfica.

Los objetivos propuestos para este trabajo, y que derivan en la obtención de una biblioteca gráfica que sustituya la actual, se corresponden con las estrategias de la empresa, ya que:

- El empleo de técnicas para la visualización eficiente de la escena, permitirá la representación de grandes escenarios sin que esto signifique lentitud en la simulación.
- La incorporación de animaciones y de las cualidades físicas y emocionales de personajes, le dará el realismo requerido a las escenas del simulador.
- El soporte a más de una biblioteca gráfica, responde a las varias tendencias que existen en el mundo, lo que ampliará el mercado.

3.2 Reglas del negocio

Las mallas de cada modelo no deben exceder una cantidad de polígonos especificados por el jefe de proyecto.

El fichero de textura a cargar para los modelos debe ser de extensión BMP o PCX, en caso de no existir el fichero en el camino indicado, ser de otro formato, estar corrupto o no tener el modelo textura no constituye un error, solo se debe indicar lo sucedido y cargar el modelo sin textura.

La posición inicial de animación de los modelos de personajes a cargar, debe ser diseñada según el estándar para los ficheros BVH (hombre erguido con manos a los lados del cuerpo).

Los roles de la animación puden ser eliminados solo cuando no están siendo utilizados por ningún personaje, por lo que es necesaria hacer la verificación antes de la eliminación.

Las animaciones no pueden ser eliminadas si pertenecen a algún rol de animación. Antes deben ser eliminadas del rol. Esta verificación debe hacerse antes de eliminar cualquier animación de la colección.

Los modelos no pueden ser elinimados si están siendo referenciados por algún nodo del grafo de escena. Esta verificación debe ser hecha antes de proceder a la eliminación.

Los personajes animados deben estar siempre en constante movimiento. Un ejemplo que ilustra esto es en el caso de una persona que se encuentre parada, ésta al menos debe girar su cabeza en diferentes direcciones cada cierto tiempo, lo que indica que no es un objeto inanimado. Excepcionalmente pudiera suceder que para determinada aplicación los personajes simularan la muerte y entonces ese sería la única variante en que estos pernamecerán inmóviles.

3.3 Modelo del dominio

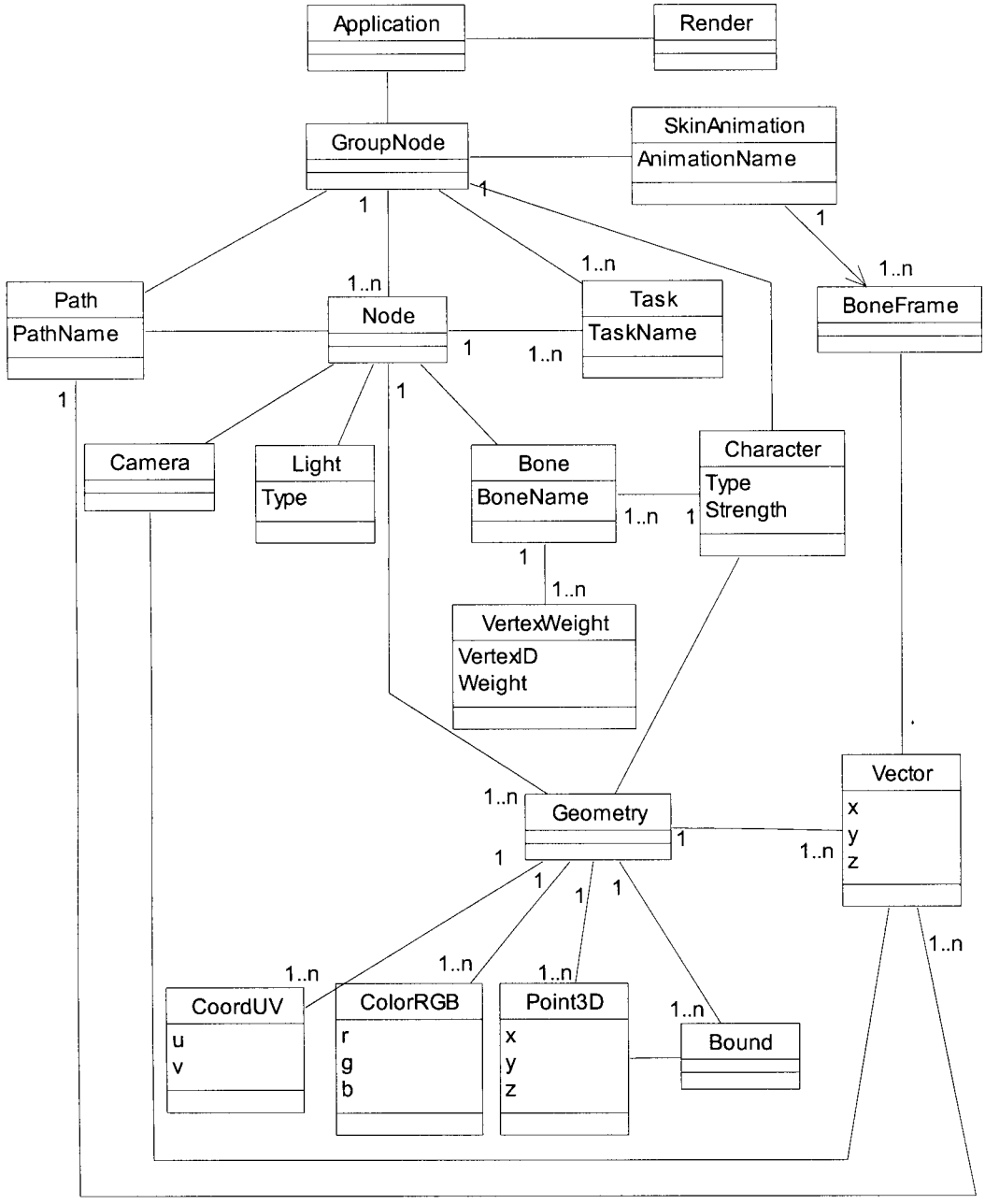

Fig. 11 Modelo del Dominio.

3.4 Captura de requisitos

A continuación se expondrán los requisitos funcionales y no funcionales del sistema.

3.4.1 Requisitos funcionales

1. Cargar datos de fichero.
2. Cargar modelo de malla con hueso.
3. Cargar modelo de malla simple.
4. Cargar animación.
5. Crear vértice.
6. Crear malla.
7. Crear hueso.
8. Crear esqueleto.
9. Crear modelo de malla con hueso.
10. Insertar modelos simples (mallas) a la colección de modelos.
11. Insertar modelos de malla con hueso a la colección de modelos.
12. Crear animaciones predefinidas, caminos y tareas.
13. Insertar animaciones predefinidas, caminos y tareas a la colección de animaciones.
14. Crear rol.
15. Crear asociaciones entre animaciones predefinidas y roles.
16. Insertar rol a la colección.
17. Destruir vértice.
18. Destruir malla.
19. Destruir hueso.
20. Destruir esqueleto.
21. Destruir modelo de malla con hueso.
22. Eliminar modelos simples (mallas) a la colección de modelos.
23. Eliminar modelos de malla con hueso a la colección de modelos.

24. Eliminar asociaciones entre animaciones predefinidas y roles.
25. Destruir rol.
26. Destruir animaciones predefinidas, caminos y tareas.
27. Eliminar animaciones predefinidas, caminos, tareas de la colección de animaciones.
28. Asignar camino y tareas a objeto o grupo de objetos.
29. Asignar animación predefinida al personaje.
30. Insertar nodos.
31. Insertar ramas.
32. Crear personaje.
33. Asociar modelo de malla con hueso al personaje.
34. Asociar rol al personaje.
35. Asignar personaje a nodo grupo.
36. Asignar hueso a nodo.
37. Crear nodo de personaje.
38. Crear nodo hueso.
39. Crear nodo de geometría.
40. Crear nodo grupo.
41. Asignar modelo simple (geometría) a nodo.
42. Crear luz.
43. Asignar luz a nodo.
44. Crear cámara.
45. Asignar cámara a nodo.
46. Crear nodo de luz.
47. Crear nodo de cámara.
48. Eliminar nodos.
49. Eliminar ramas.
50. Destruir nodo.
51. Destruir luz.
52. Destruir cámara.
53. Destruir personaje.

54. Modificar propiedades del personaje (Ej. estado físico).
55. Modificar propiedades de la luz.
56. Modificar propiedades de la cámara.
57. Actualizar estado geométrico (matrices de transformación) de un nodo.
58. Actualizar estado geométrico de un nodo manualmente.
59. Actualizar volumen de colisión de objeto o grupo de objetos dibujables.
60. Actualizar estados de *render* de objeto o grupo de objetos dibujables.
61. Actualizar estados de render manualmente.
62. Modificar estado geométrico de la luz.
63. Modificar estado geométrico de la cámara.
64. Determinar si un objeto está en el volumen de visión de la cámara.
65. Calcular distancia de un punto a la cámara.
66. Aplicar algoritmo de eliminación de triángulos por niveles de detalles.
67. Deformar la malla del personaje según animación del esqueleto.
68. Texturizar malla por niveles de detalles.
69. Aplicar estados geométricos (aplicar matrices de rotación, traslación y escalado).
70. Aplicar estados de *render.*
71. Inicializar aplicación.
72. Controlar el tiempo de la aplicación.
73. Hacer ciclo de actualización de escena.

3.4.2 Requisitos no funcionales

- **Usabilidad:** Los futuros usuarios del sistema serán programadores con conocimientos básicos de animación y de la terminología afín. El producto debe estar concebido para que el usuario piense en qué desea hacer y no cómo hacerlo, por lo que éste requerimiento debe estar presente en alto grado en el producto final.

- **Rendimiento:** Como aplicación de tiempo real, debe tener alto grado de velocidad de procesamiento o cálculo, tiempo de respuesta y de recuperación, y disponibilidad.
- **Soporte:** En una versión inicial deberá ser compatible con la plataforma Windows, pero debe estar preparado para que con rápidas modificaciones pueda migrar para Linux.
- **Legales:** Se regirá por las normas ISO 9000.
- **Software:** Sistema operativo Windows.
- **Hardware:** Compatibilidad con tarjetas gráficas de la familia NVIDIA (Geforce 3, Geforce 4, FX 5200).
- **Diseño e implementación**: Debe utilizar transparentemente la biblioteca gráfica OpenGL y DirectX, en su primera versión, y ser adaptable a trabajar con otras bibliotecas. Se harán llamadas a dichas bibliotecas desde Leguaje C++. Se regirá por la filosofía de Programación Orientada a Objetos.

3.5 Modelo de casos de usos del sistema

En esta sección se reconocen los posibles actores del sistema a desarrollar y se conciben, a través de la agrupación de los requisitos funcionales anteriormente hallados, los posibles resultados de valor que le pueda brindar a sus actores, o lo que es lo mismo, los casos de uso del sistema.

Además, se seleccionan los casos de uso correspondientes al primer ciclo de desarrollo para hacerles sus especificaciones textuales en formato expandido.

Actor del sistema

Tabla 2 Actor del sistema

Actores	Justificación
Programador	Es el que se beneficiará con las funcionalidades que brinda la biblioteca de clases, a grosso modo: cargar desde ficheros, actualizar los datos y ejecutar el ciclo de la escena.

Casos de uso de sistema

1. Cargar ficheros.
2. Cargar modelo.
3. Cargar animación.
4. Crear entidad geométrica.
5. Adicionar modelo a la colección.
6. Adicionar animación a la colección.
7. Actualizar roles.
8. Eliminar modelo de la colección.
9. Eliminar animación de la colección.
10. Asignar animación a objeto o grupo de objetos.
11. Insertar nodo.

12. Crear personaje.
13. Crear instancia de modelo simple a la escena.
14. Crear luz o cámara.
15. Eliminar nodo.
16. Destruir nodo.
17. Modificar propiedades del personaje.
18. Modificar propiedades de luz o cámara.
19. Actualizar estado geométrico.
20. Actualizar estado geométrico manualmente.
21. Actualizar volumen de colisión.
22. Actualizar estados de *render*.
23. Actualizar estados de *render* manualmente.
24. Enviar a *render*.
25. Inicializar y actualizar escena.

Paquetes de casos de uso del sistema

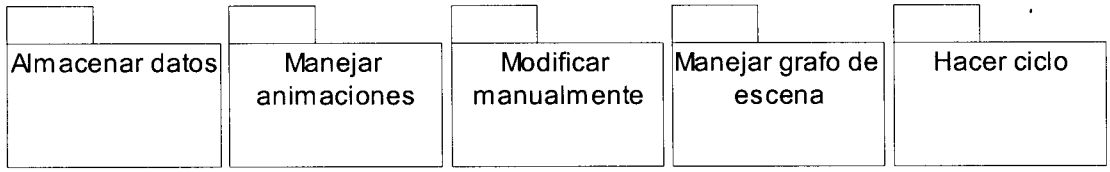

Fig. 12 Paquetes de casos de uso del sistema

El primer ciclo de desarrollo tiene como objetivo visualizar un modelo simple cargado desde un fichero, para lo cual se decidió desarrollar las funcionalidades que brindan los casos de uso: cargar ficheros, cargar modelo, crear geometría, adicionar modelo a la colección, eliminar modelo de la colección, insertar nodo, crear instancia de modelo simple a la escena, crear luz o cámara, eliminar nodo, destruir nodo, actualizar estado geométrico manualmente, enviar a *render*, e inicializar y actualizar escena. Éstos además son el soporte para los posteriores

ciclos de desarrollo donde se cargarán modelos más complejos, se ejecutarán animaciones y se optimizará la visualización de la escena.

3.5.1 Paquete "Almacenar datos"

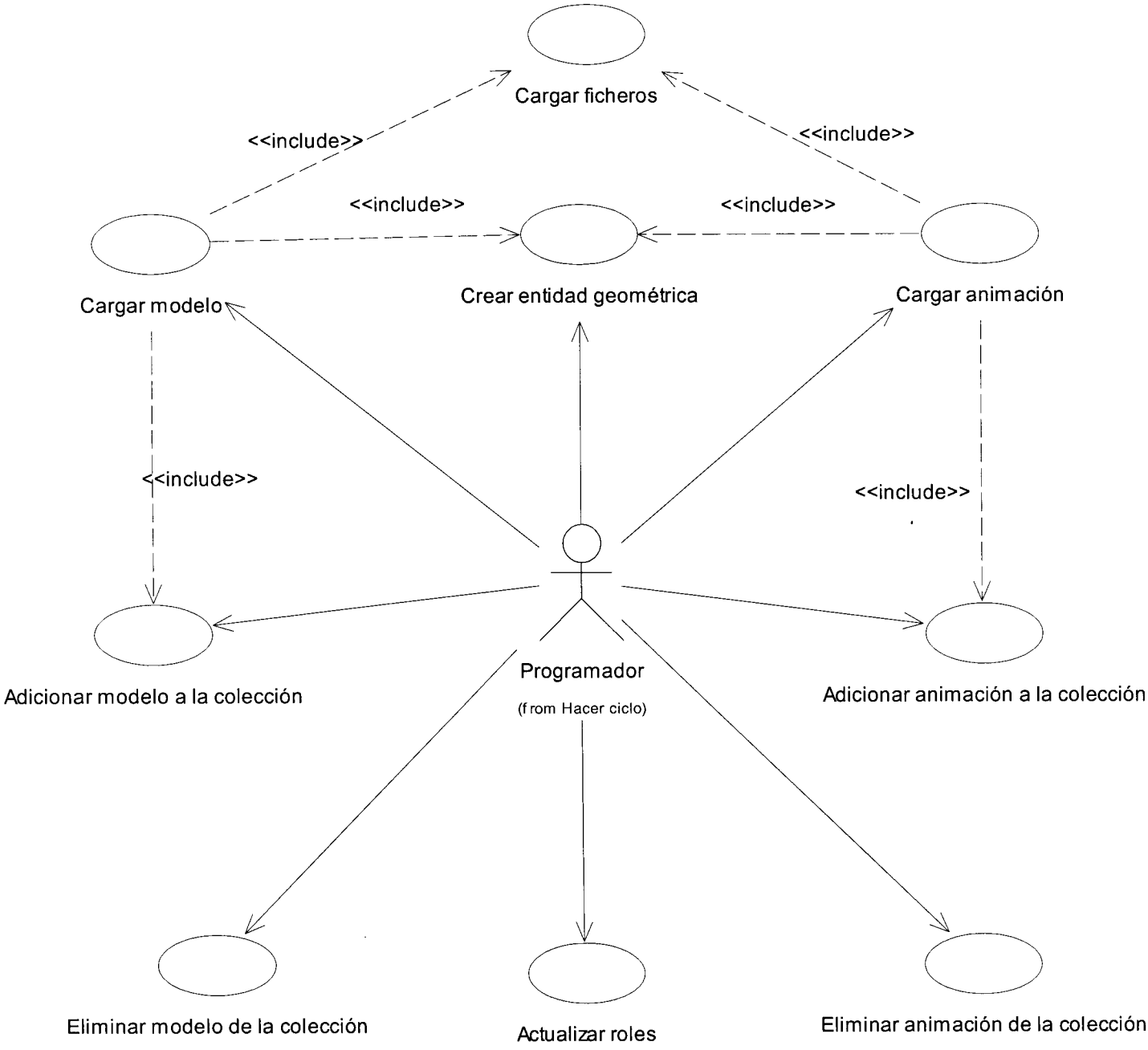

Fig. 13 Paquete CU "Almacenar datos".

Especificación de los casos de uso del paquete en formato expandido

Tabla 3 CU Cargar ficheros.

Nombre del caso de uso	**Cargar ficheros.**
Actores	
Propósito	Cargar información almacenada en ficheros.
Resumen Se inicia cuando el CU "Cargar modelo" solicita cargar datos de un fichero. Entonces se procede a almacenar en un *buffer* de memoria la cantidad de datos desde la posición indicada por el CU solicitante. En caso de que la lectura sea fallida se le informa al CU iniciador. El caso de uso finaliza cuando se carga la información en el *buffer* u ocurre algún error.	
Referencias	R1
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
	1- Se abre el fichero a cargar. En caso de ocurrir error se detiene la ejecución y se devuelve el error.
	2- Se prepara el fichero para leer a partir de una posición. En caso de ocurrir error se detiene la ejecución y se devuelve el error.
	3- Se lee la cantidad de *bytes* indicados y se almacenan en un *buffer.* En caso de ocurrir error se detiene la ejecución y se devuelve el error.
	4- Se cierra el fichero.

Tabla 4 CU Cargar modelo.

Nombre del caso de uso	**Cargar modelo.**
Actores	Programador
Propósito	Interpretar el *buffer* de datos del fichero y almacenarlos en las estructuras de datos de los modelos.
Resumen Se inicia cuando el Programador pasa una dirección de fichero. De ser válida la dirección se llama al CU "Cargar ficheros", el cual llena los *buffers*. Entonces se crean las estructuras necesarias para almacenar el modelo, y se llama al CU "Adicionar modelo a la colección". De ocurrir algún error en medio del proceso se informa al Programador. El caso de uso finaliza cuando se carga en memoria el modelo u ocurre algún error.	
Referencias	R2, R3. CU1, CU4, CU5 (*include*)
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1- Solicita cargar un fichero con un camino dado.	
	2- Se verifica la existencia del fichero y si este tiene una extención válida.
	3- Se crea un controlador de ficheros para que haga la carga de la información en un *buffer.*
	4- Se carga en encabezamiento del fichero. En caso de error ir al paso 8.
	5- Se carga los datos del modelo con la información del encabezamiento del fichero. En caso de error ir al paso 8.
	6- Se crea una malla con los datos geométricos del modelo.
	7- Se crea y adiciona a la colección un nuevo modelo pasando la malla creada y el nombre

	del modelo que venía en fichero.
	8- Se devuelve la validez de las operaciones.
Poscondiciones	Creados nuevos objetos de escena.

Nota: en el caso de uso anterior no se tiene en cuenta para este ciclo de desarrollo, el caso en el que sea un modelo de malla con huesos.

Tabla 5 CU Crear entidad geométrica.

Nombre del caso de uso	**Crear entidad geométrica.**
Actores	Programador
Propósito	Crear una malla y todos objetos geométricos que la componen.
Resumen Se inicia cuando el actor o el caso de uso "Cargar Modelo" solicita crear una entidad geométrica, entonces, el caso de uso finaliza cuando ésta es creada.	
Referencias	R5, R6. CU1, CU5 (*include*)
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
Sección: Crear punto.	
1- Se solicita la creación de un punto pasándole sus tres componentes.	
	2- Se crea un punto.
Sección: Crear color.	
1- Se solicita la creación de un color pasándole los valores R, G y B.	
	2- Se crea un color.

Sección: Crear vector.	
1- Se solicita la creación de un vector pasándole las tres componentes.	
	2- Se crea un vector.
Sección: Crear volumen frontera.	
1- Se solicita la creación de un volumen frontera.	
	2- Se crea un volumen frontera.
Sección: Crear coordenada de textura.	
1- Se solicita la creación de una coordenada de textura pasándole los valores u, v.	
	2- Se crea la coordenada de textura.
Sección: Crear malla.	
1- Se solicita la creación de una malla pasándole la cantidad de vértices, la lista de vértices, la cantidad de puntos, la lista de puntos, la lista de normales, dos listas de coordenadas de texturas, la cantidad de volúmenes fronteras, y la lista de volúmenes.	
	2- Se crea la malla.
Poscondiciones	Creados objetos geométricos.

Tabla 6 CU Adicionar modelo a la colección.

Nombre del caso de uso	**Adicionar modelo a la colección.**
Actores	Programador
Propósito	Adicionar un muevo modelo.
Resumen: Se inicia cuando el actor o el CU "Cargar modelo" invoca a adicionar modelo, pasando la lista de mallas y el nombre del modelo. Entonces se crea un modelo con esos parámetros y finaliza cuando se inserta en la lista con un ID asignado.	
Referencias	R7-R11.
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1- Solicita la adición de un modelo, dados sus mallas y su nombre.	
	2- Se crea un modelo con los parámetros indicados.
	3- Se le asigna un ID al modelo.
Poscondiciones	Nuevo elemento en la lista de modelos.

Nota: en el caso de uso anterior no se tiene en cuenta para este ciclo de desarrollo, el caso en el que sea un modelo de malla con huesos.

Tabla 7 CU Eliminar modelo de la colección.

Nombre del caso de uso	**Eliminar modelo de la colección.**
Actores	Programador
Propósito	Actualizar la estructura y contenido del grafo de la escena.
Resumen Se inicia cuando el actor invoca eliminar un modelo pasándosele el nombre o el ID. Entonces se localiza el modelo y si no está asociado a ningún nodo de la escena se elimina. Finalmente se devuelve si la acción tuvo validez. El caso	

de uso culmina cuando es eliminado el modelo u ocurre algún error.	
Referencias	R17-R23
Curso normal de los eventos	
Acción del actor	Respuesta del sistema
1- Solicita eliminar un modelo de la colección.	
	2- Se busca el modelo. Si no está ir al paso 5.
	3- Se verifica si no está siendo utilizado. En caso contrario ir al paso 5.
	4- Se elimina el modelo y los objetos que lo componen.
	5- Se devuelve la validez de la operación.
Precondiciones	Debe existir al menos un modelo en la lista.
Poscondiciones	Queda decrementada la lista de modelos.

Nota: en el caso de uso anterior no se tiene en cuenta para este ciclo de desarrollo, el caso en el que sea un modelo de malla con huesos.

3.5.2 Paquete "Manejar grafo de escena"

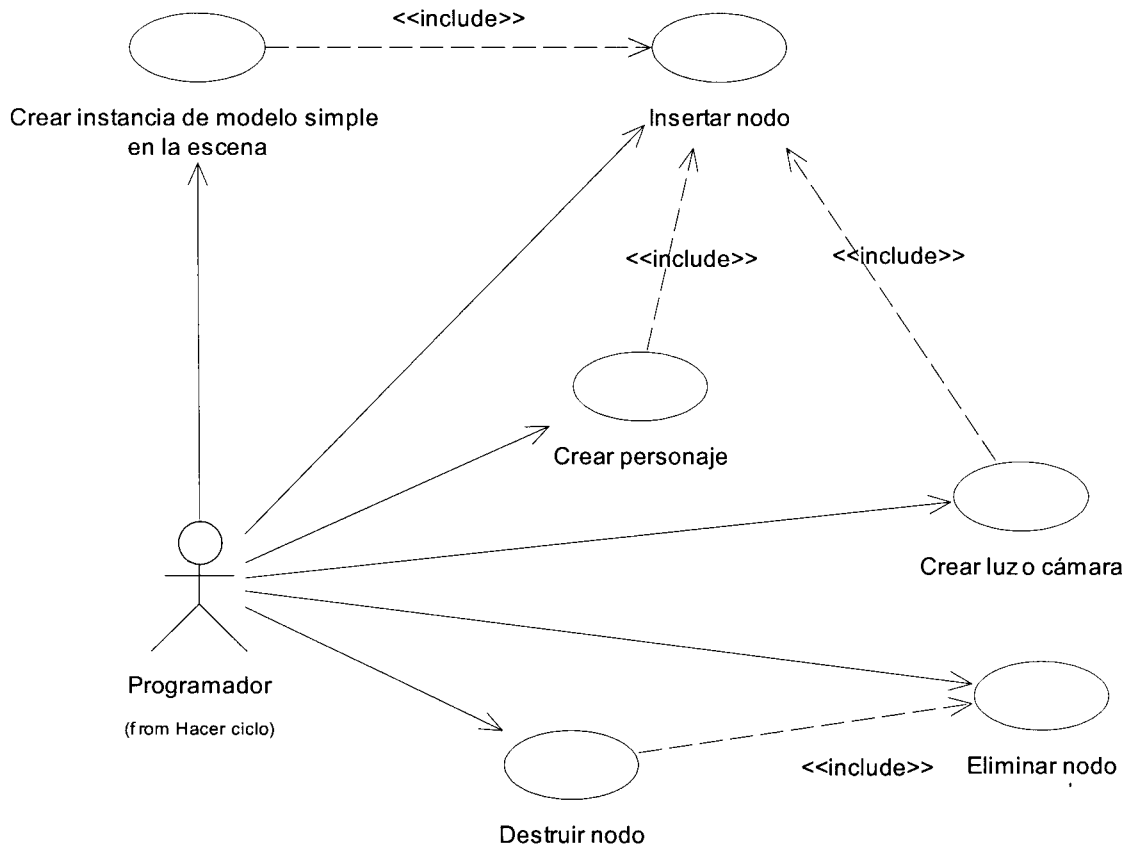

Fig. 14 Paquete CU "Manejar grafo de la escena".

Especificación de los casos de uso del paquete en formato expandido

Tabla 8 CU Insertar nodo.

Nombre del caso de uso	**Insertar nodo.**
Actores	Programador
Propósito	Insertar un nodo existente en el grafo una posición del grafo de la escena.
Resumen Se inicia cuando el Programador invoca la inserción de un nodo en otro, pasando el nodo a insertar. Finaliza cuando éste es adicionado a la lista del nodo padre y se le crea el vínculo al padre.	
Referencias	R30-R31.
Curso normal de los eventos	
Acción del actor	Respuesta del sistema
1- Solicita, teniendo un nodo grupo, la inserción de un nodo hijo pasado como parámetro.	
	2- Se adiciona el nodo a la lista de hijos del grupo.
	3- Se crea un enlace del nodo insertado a su padre.
Precondiciones	Debe existir al menos un nodo en el grafo.
Poscondiciones	Queda incrementado el grafo.

Tabla 9 CU Crear instancia de modelo simple en la escena.

Nombre del caso de uso	**Crear instancia de modelo simple en la escena.**
Actores	Programador
Propósito	Crear una instancia de modelo simple.
Resumen Se inicia cuando el Programador, para adicionar un modelo simple a la escena, hace una búsqueda de un modelo y del nodo grupo al que pertenecerá, y en caso de encontrarlos, invoca la creación de una instancia del modelo en la escena, dados el modelo y el nodo encontrados. Las inserciones de nodos se hacen utilizando al CU "Insertar nodo". En caso de no ser válida alguna de las operaciones se devolverá el error. El caso de uso finaliza cuando se crea la instancia del modelo u ocurre algún error.	
Referencias	R39-R41. CU11(*include*)
Curso normal de los eventos	

Acción del actor	Respuesta del sistema
1- Solicita la búsqueda de un modelo dado su ID o su nombre.	
	2- Se busca el modelo; en caso de existir, se devuelve la posición; sino, ir al paso 15.
3- Solicita el modelo en la posición encontrada.	
	4- Se devuelve el modelo. En caso de error, ir al paso 15.
5- Solicita la búsqueda de un nodo dado su ID, o dado su nombre, o dado el ID o el nombre de un objeto, y su tipo.	
	6- Se busca el nodo. En caso de no

	existir, o existir y no ser nodo grupo, ir al paso 15.
7- Solicita la inserción del modelo en un nodo grupo, dado el modelo y el nodo.	
	8- Si el modelo tiene una única malla, ir a sección 1; si tiene varias mallas, ir a sección 2.
Sección 1: Malla única.	
	9- Se crea un nodo geometría y se le asigna la malla.
	10- Se inserta el nodo geometría en el nodo grupo indicado.
Sección 2: Múltiples mallas.	
	11- Se crea un nodo grupo.
	12- Se crean nodos geometría y se asigna una malla a cada uno.
	13- Se insertan los nodos geometría en el nodo grupo creado.
	14- Se inserta el nodo grupo creado en el nodo grupo indicado.
Fin de sección.	
	15- Se devuelve la validez del proceso.
Precondiciones	Deben existir al menos un modelo simple.
Poscondiciones	Queda incrementado el grafo de la escena.

Tabla 10 CU Crear luz o cámara.

Nombre del caso de uso	**Crear luz o cámara.**
Actores	Programador
Propósito	Crear una luz o una cámara para la escena.
Resumen Se inicia cuando el Programador crea una luz o una cámara para insertarla en escena, entonces busca el nodo al que pertenecerá la luz o cámara creada, e invoca la inserción de la luz o cámara en el nodo buscado. Se inserta el nodo en el grafo de la escena utilizando al CU "Insertar nodo". En caso de no ser válida alguna de las operaciones se devolverá un error. El caso de uso finaliza cuando se crea la luz o cámara u ocurre algún error.	
Referencias	R42-R47. CU11(*include*)
Curso normal de los eventos	
Acción del actor	Respuesta del sistema
Sección 1: Crear cámara.	
1- Solicita la búsqueda de un nodo dado su ID, o dado su nombre, o dado el ID o el nombre de un objeto, y su tipo.	
	2- Se busca el nodo; en caso de no existir, o existir y no ser nodo grupo, ir al paso 8.
3- Solicita la creación de una cámara, indicando si será perspectiva o no.	
	4- Se crea la cámara.
5- Solicita la inserción de la cámara creada, en el nodo buscado.	
	6- Se crea un nodo cámara y se le

	asigna la cámara creada.
	7- Se inserta el nodo cámara en el nodo grupo indicado.
Sección 2: Crear luz.	
1- Solicita la búsqueda de un nodo dado su ID, o dado su nombre, o dado el ID o el nombre de un objeto, y su tipo.	
	2- Se busca el nodo; en caso de no existir, o existir y no ser nodo grupo, ir al paso 8.
3- Solicita la creación de una luz, pasando los parámetros necesarios según el tipo de luz que desee crear.	
	4- Se crea la luz.
5- Solicita la inserción de la luz creada, en el nodo buscado.	
	6- Se crea un nodo luz y se le asigna la luz creada.
	7- Se inserta el nodo luz en el nodo grupo indicado.
Fin de sección.	
	8- Se devuelve la validez del proceso.
Poscondiciones	Queda incrementado el grafo de la escena.

Tabla 11 CU Eliminar nodo.

Nombre del caso de uso	**Eliminar nodo.**
Actores	Programador
Propósito	Eliminar un nodo del grafo de la escena
Resumen Se inicia cuando el Programador invoca a desenlazar un nodo determinado de su padre. Entonces el padre le quita al nodo la referencia a él y lo elimina de su lista. El caso de uso finaliza cuando se elimina el nodo y se devuelve como resultado.	
Referencias	R48-R49
Curso normal de los eventos	
Acción del actor	Respuesta del sistema
1- Solicita, teniendo un nodo grupo, la eliminación de un nodo hijo pasado como parámetro.	
	2- Se elimina el nodo de la lista de hijos del grupo.
	3- Se elimina el enlace del nodo eliminado a su padre.
Precondiciones	Deben existir al menos dos nodos en el grafo de la escena.
Poscondiciones	Queda decrementado el grafo de la escena.

Tabla 12 CU Destruir nodo.

Nombre del caso de uso	**Destruir nodo.**
Actores	Programador
Propósito	Destruir nodo del grafo de la escena.
Resumen El caso de uso inicia cuando el Programador busca un nodo y lo manda a destruir. Entonces, si el nodo contiene una luz o una cámara, se elimina el nodo y el objeto; si contiene una geometría, se destruye el nodo y se descuenta la cantidad de propietarios del modelo de la geometría; si contiene un personaje, se descuenta la cantidad de propietarios del modelo, y se destruyen los nodos huesos del personaje, el personaje y el nodo personaje; y si el nodo es agrupador, se destruyen sus hijos y el nodo grupo. Finalmente se devuelve el resultado de la operación. El caso de uso finaliza con la destrucción del nodo o con la ocurrencia de algún error.	
Referencias	R50-R53
Curso normal de los eventos	
Acción del actor	Respuesta del sistema
1- Solicita la búsqueda de un nodo dado su ID, o dado su nombre, o dado el ID o el nombre de un objeto, y su tipo.	
	2- Se busca el nodo; en caso de no existir, o existir y no ser nodo grupo, ir al paso 19.
3- Solicita destuir un nodo pasado como parámetro.	
	4- Si el tipo de nodo es cámara, ir a sección 1; si es luz, ir a sección 2; si es geometría, ir a sección 3; si es

	personaje, ir a sección 4; si es grupo, ir a sección 5.
Sección 1: Destruir nodo cámara.	
	5- Se elimina el nodo del grafo.
	6- Se destruye la cámara.
	7- Se destruye el nodo.
Sección 2: Destruir nodo luz.	
	8- Se elimina el nodo del grafo.
	9- Se destruye la luz.
	10- Se destruye el nodo.
Sección 3: Destruir nodo geometría.	
	11- Se elimina el nodo del grafo.
	12- Se obtiene el ID de la malla.
	13- Se destruye el nodo.
	14- Se busca el modelo al que pertenece la malla.
	15- Se decrementa la cantidad de propietarios del modelo.
Sección 5: Destruir nodo grupo.	
	16- Se elimina el nodo del grafo.
	17- Se destruyen los nodos hijos.
	18- Se destruye el nodo.
Fin de sección.	
	19- Se devuelve la validez del proceso.
Precondiciones	Debe existir al menos un nodo.
Poscondiciones	Se decrementa el grafo de la escena

Nota: en el caso de uso anterior no se tiene en cuenta para este ciclo de desarrollo, el caso en el que sea un nodo personaje.

3.5.3 Paquete "Modificar manualmente"

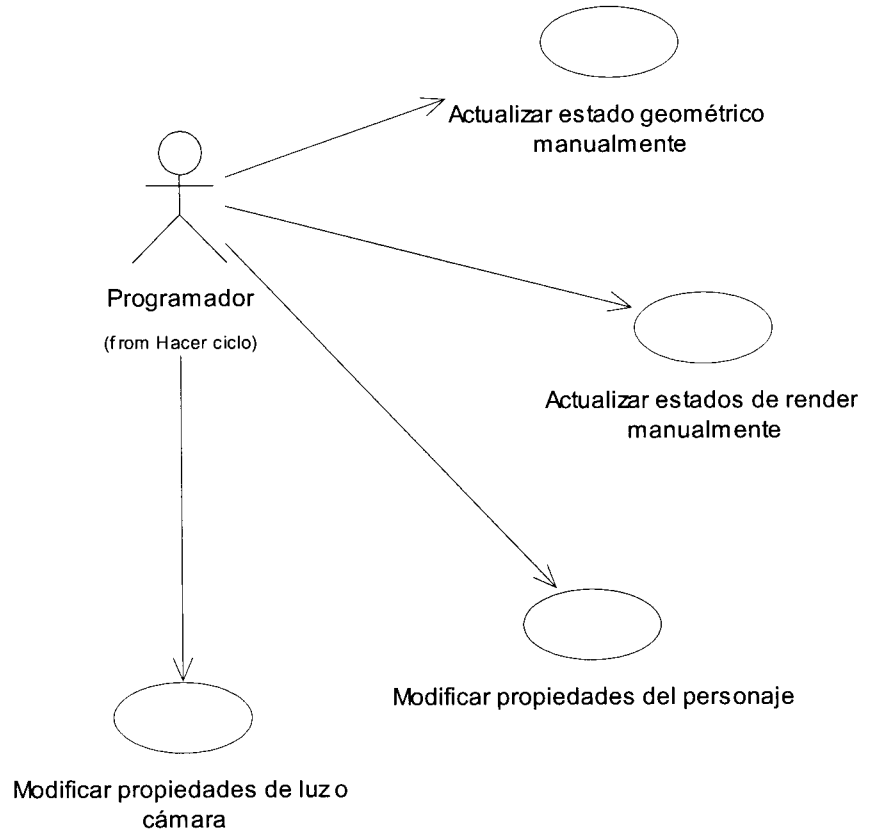

Fig. 15 Paquete CU "Modificar manualmente".

Especificación de los casos de uso del paquete en formato expandido

Tabla 13 CU Actualizar estado geométrico manualmente.

Nombre del caso de uso	**Actualizar estado geométrico manualmente.**
Actores	Programador
Propósito	Actualizar directamente las matrices de transformación locales de un nodo (de contenido o agrupador).
Resumen: El caso de uso puede ser invocado de dos maneras: para actualizar todas las matrices de transformación del nodo, o para actualizar una de ellas. En ambos casos se inicializan las matrices y se actualizan con los nuevos parámetros,	

dejándose una indicación de actualización manual. Finalmente se devuelve la validez de la operación. El caso de uso concluye con la actulización de las matrices o con la ocurrencia de algún error.	
Referencias	R58
Curso normal de los eventos	
Acción del actor	Respuesta del sistema
1- Solicita modificar el estado geométrico de un nodo, suministrando un identificador del nodo y todas las matrices de transformación (ver sección 1), o un identificador del nodo, una matriz de transformación y el tipo de transformación a realizar (ver sección 2).	
	2- Se busca el nodo a modificar, en caso de no exisir, ir al paso 9
Sección 1: Actualizar todas las matrices.	
	3- Se sustituyen todas las matrices de transformación del nodo.
Sección 2: Actualizar una matriz.	
	4- Se inicializan las matrices de transformación del nodo Si el tipo de transformación es "rotar", ir a sección 2.1; si es "escalar", ir a sección 2.2; si es "trasladar", ir a sección 2.3.
Sección 2.1: Rotar.	
	5- Se sustituye la matriz de rotación.
Sección 2.2: Escalar.	
	6- Se sustituye la matriz de escala.

Sección 2.3: Trasladar.	
	7- Se sustituye la matriz de traslación.
Fin de sección.	
	8- Se pone una indicación de actualización manual.
	9- Se devuelve la validez de la operación.
Precondiciones	Debe estar creado el grafo de la escena.
Poscondiciones	Quedan actualizadas las matrices de transformación.

3.5.4 Paquete "Hacer ciclo"

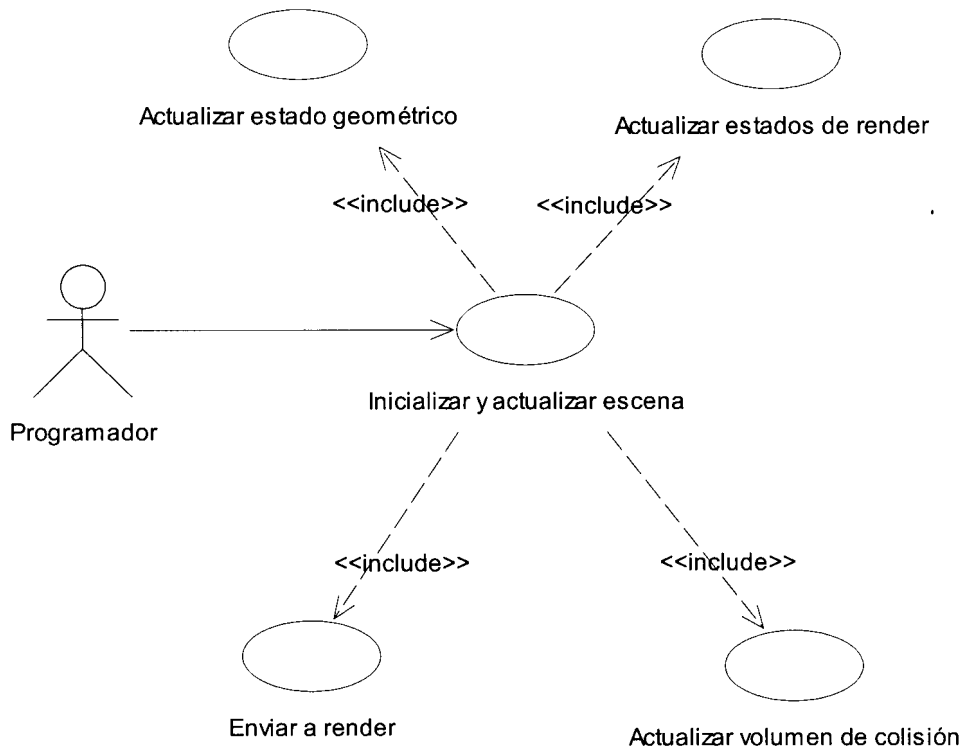

Fig. 16 Paquete CU "Hacer ciclo".

Especificación de los casos de uso del paquete en formato expandido

Tabla 14 CU Enviar a render.

Nombre del caso de uso	**Enviar a *render*.**
Actores	
Propósito	Preparar el objeto de un nodo para ser enviado al *render*, y enviarlo.
Resumen El caso de uso es invocado por el CU "Inicializar y Actualizar escena". En el caso de uso se preparan las condiciones para enviar un objeto al *renderer:* en el caso de que el nodo contenga una cámara o una luz, se actualiza el estado geométrico del objeto; en el caso de que contenga una geometría y esté en el volumen de visión de la cámara, se le optimiza la cantidad de polígonos y se envía al renderer con los estados geométricos y de *render*; donde se dibuja con las características dadas; y en el caso de que sea un nodo de agrupación y esté en el volumen de visión de la cámara, se envían a *render* sus nodos hijos. El caso de uso concluye con el envío a *render* del contenido del nodo.	
Referencias	R62-R70
Curso normal de los eventos	
Acción del actor	Respuesta del sistema
Sección 1: Nodo Cámara	
	1- Se actualiza la posición y orientación de la cámara con las matrices de transformación del nodo.
Sección 2: Nodo luz	
	2- Si la luz es direccional, ir a sección 3; si es puntual, ir a sección 4; si es *spot*, ir a sección 5.
Sección 3: Luz direccional	
	3- Se actualiza la orientación de la luz con la matriz de rotación del nodo.

Sección 4: Luz puntual	
	4- Se actualiza la posición de la luz con la matriz de traslación del nodo.
Sección 5: Luz *spot*	
	5- Se actualiza la posición y orientación de la luz con las matrices de transformación del nodo.
Sección 6: Nodo geometría	
	6- Se obtiene la cámara activa.
	7- Se comprueba si la malla está en el volumen de visión de la cámara activa. Si no está, ir al paso 16.
	8- Se calcula la distancia de la malla a la cámara activa.
	9- Se busca el nivel de detalle que tiene definida la malla para la la distancia calculada a la cámara.
	10- Se reducen los polígonos de la malla según el nivel de detalle definido.
	11- Se le pasa al *renderer* la malla con el estado geométrico (matrices de transformación) y los estados de render.
	12- Se dibuja la malla con los estados indicados.
Sección 7: Nodo grupo	
	13- Se obtiene la cámara activa.
	14- Se comprueba si el volumen frontera del nodo está en el volumen de visión de la cámara. Si no está, ir al paso 16.
	15- Se envian a *render* cada uno de los hijos del nodo grupo.
Fin de sección	
	16- Fin del proceso.

Precondiciones	Debe estar creado el grafo de la escena.

Nota: en el caso de uso anterior no se tiene en cuenta para este ciclo de desarrollo, el caso en el que el nodo contiene un personaje.

Tabla 15 CU Inicializar y actualizar escena.

Nombre del caso de uso	**Inicializar y actualizar escena.**
Actores	Programador
Propósito	Inicializar la aplicación y hacer el ciclo de actualización de la escena.
Resumen El caso de uso es iniciado por el programador, y lo accede para inicializar la aplicación (preparar el *timer* y la cámara activa), o para hacer un ciclo de actualización de la escena (calcular el tiempo de la aplicación, enviar al *render* los nodos del grafo, y esperar el tiempo de dibujo de un *frame*). El caso de uso finaliza con la inicialización o la realización de un ciclo.	
Referencias	R71-R73
Curso normal de los eventos	
Acción del actor	Respuesta del sistema
Sección 1: Inicializar escena.	
1- El actor solicita inicializar la escena.	
	2- Se reinicia el *timer* (tiempo=0).
	3- Se posiciona y se orienta la cámara.
	4- Se establecen los planos del *frustum* de la cámara.

	5- Se activa la cámara.
Sección 2: Hacer ciclo de escena.	
1- El actor solicita hacer un ciclo de escena.	
	2- Se obtiene el tiempo de inicio del ciclo.
	3- Se envía a render el grafo de la escena.
	4- Se obtiene el tiempo de fin del ciclo.
	5- Se espera a que termine el tiempo asignado para cada ciclo.
	6- Se actualizan los *clics* del *timer* (cantidad de ciclos desde el inicio de la aplicación).
Precondiciones	Debe estar creado el grafo de la escena.

Nota: en el caso de uso anterior no se tiene en cuenta para este ciclo de desarrollo, las llamadas a los casos de uso de actualizaciones ("Actualizar estado geométrico", "Actualizar estados de render", y "Actualizar volumen de colisión").

3.5.5 Paquete "Manejar animaciones"

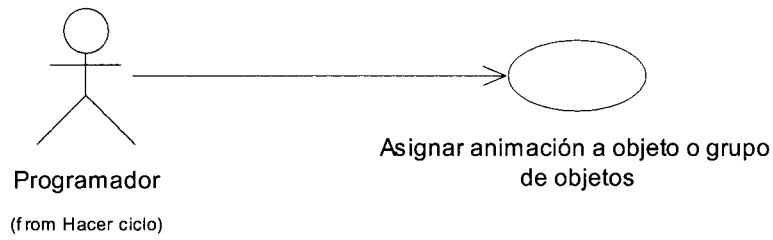

Fig. 17 Paquete CU "Manejar animaciones".

Conclusiones

En el presente capítulo se definió qué es exactamente lo que espera el usuario con este sistema. Para ello quedaron establecidos los requisitos funcionales de éste, y descritos los casos de uso que les permitirán a su futuro usuario obtener los resultados esperados por el cliente.

4

Capítulo 4 Análisis y diseño del Sistema

Introducción

La primera parte del capítulo encierra los diagramas de clases de análisis del sistema propuesto. Su objetivo es brindar una primera visión de las posibles clases del diseño a un alto nivel, pero donde se dejan ver sus responsabilidades y relaciones, aún sin el mayor rigor, entre ellas.

A continuación se presentan los diagramas de clases por paquetes como resultado del refinamiento de las etapas anteriores. Se presentan además los diagramas de secuencia de la realización de los casos de uso, que intervendrán en el primer ciclo de desarrollo del proyecto.

4.1 Diagrama de Paquetes de clases de análisis

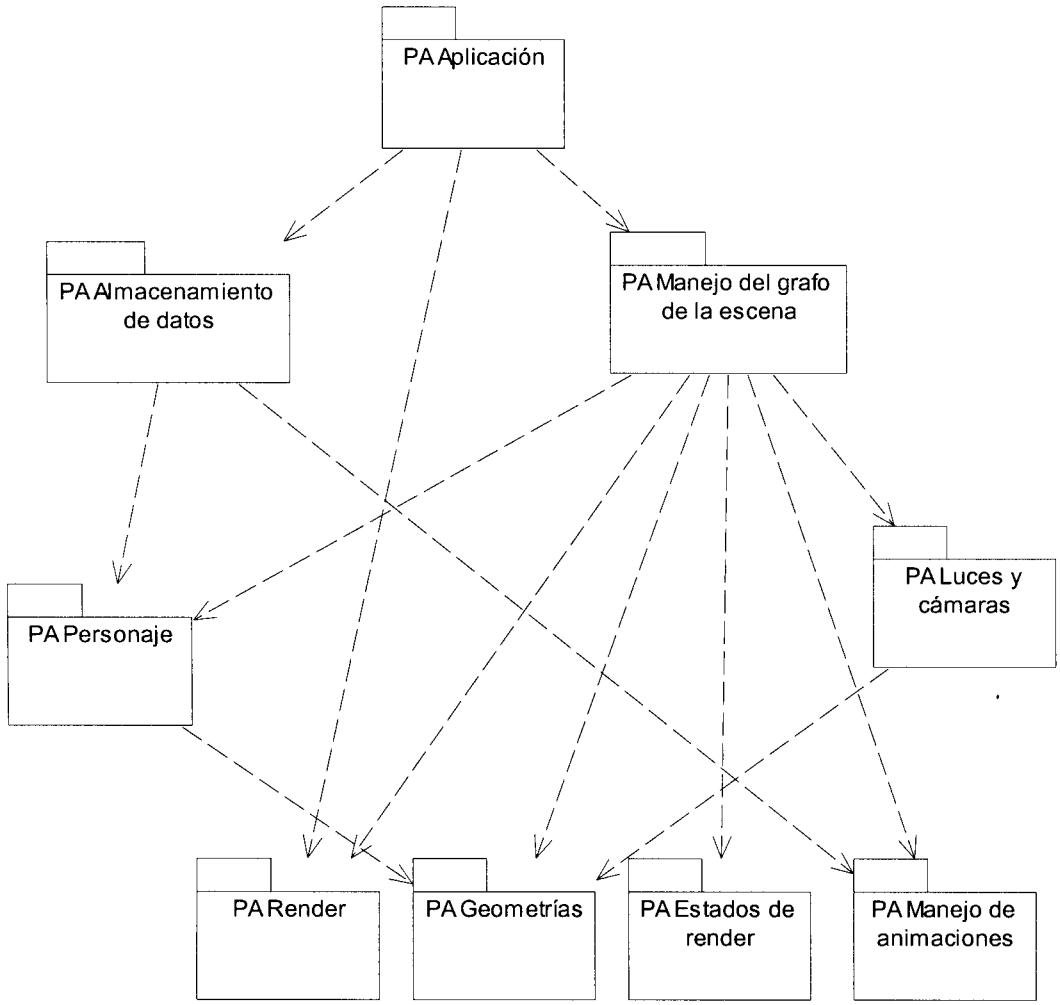

Fig. 18 Diagrama de Paquetes de Clases de Análisis.

El contenido de los paquetes de clases de análisis se muestra en el anexo "Diagramas de clases de análisis".

4.2 Diagramas de clases de diseño

Antes de pasar a mostrar los diagramas de clases, se aclararán tres cuestiones importantes para la comprensión de éstos:

1- La nomenclatura utilizada para los diagramas de clases, se explica en el epígrafe "Estándares de codificación" del siguiente capítulo.
2- En la realización de este proyecto, y a petición del cliente, se generó el código con la herramienta utilizada para el proceso de desarrollo del sistema (Rational Rose Enterprise Edition, 2003), la cual brinda la posibilidad de generar los métodos de acceso a miembros de clases ("gets" y "sets"), constructores por defecto, constructores de copia y destructores, aun cuando no se hallan incluido en las especificaciones de las clases; por tanto, y para evitar su repetición, se omite en las clases los métodos antes mencionados. En algunos casos se incluyen algunos constructores, pero no son los "por defecto".
3- Existe una clase abstracta "SP_CObject", de la cual heredan las demás clases, algunas directamente y otras indirectamente. Para evitar la complejidad de la representación de estas relaciones de herencia, se omitieron en los diagramas. Las clases que heredan directamente de "SP_CObject", y de las cuales heredan todas las demás, son:

SP_CAlphaState
SP_CAnimCollection
SP_CApplication
SP_CBoneFrame
SP_CBound
SP_CColorRGB
SP_CController
SP_CCullState
SP_CFileController
SP_CFogState
SP_CImage
SP_CInterpreter
SP_CLightState
SP_CMaterialState
SP_CMatrix3
SP_CMatrix4
SP_CModelCollection
SP_CNode
SP_CPath
SP_CPoint2D
SP_CPolygonOffSetState
SP_CPolygonReductor
SP_CRenderer
SP_CRenderState
SP_CRole
SP_CSceneObject
SP_CShadeState
SP_CSingleModel
SP_CSkeleton
SP_CSkinAnimation
SP_CTask
SP_CTexture
SP_CTextureState
SP_CTimer
SP_CTransformator
SP_CTransformData
SP_CVector3
SP_CVertexColorState
SP_CVertexWeight
SP_CZBufferState

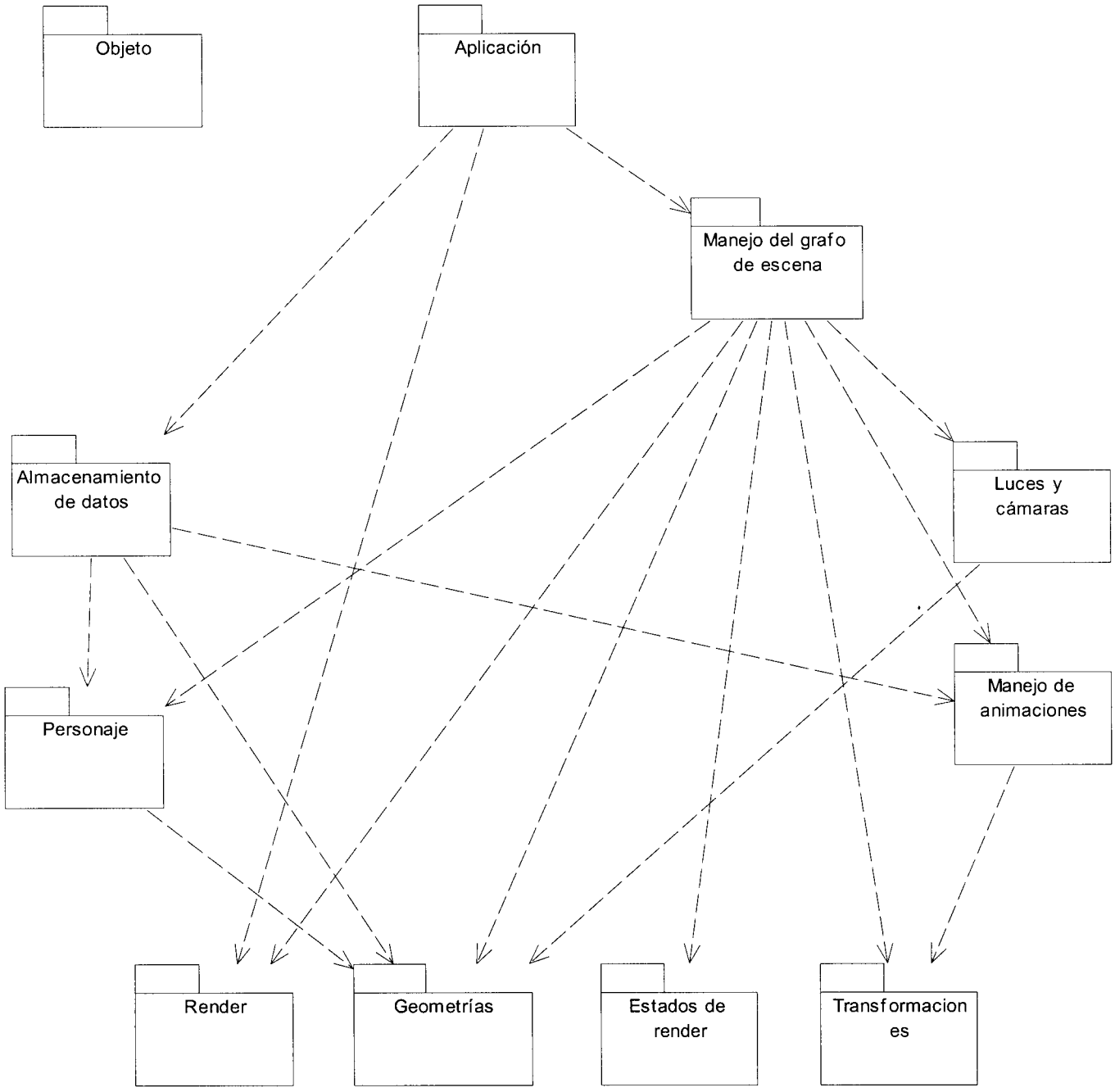

Fig. 19 Diagrama de Paquetes de Clases de Diseño.

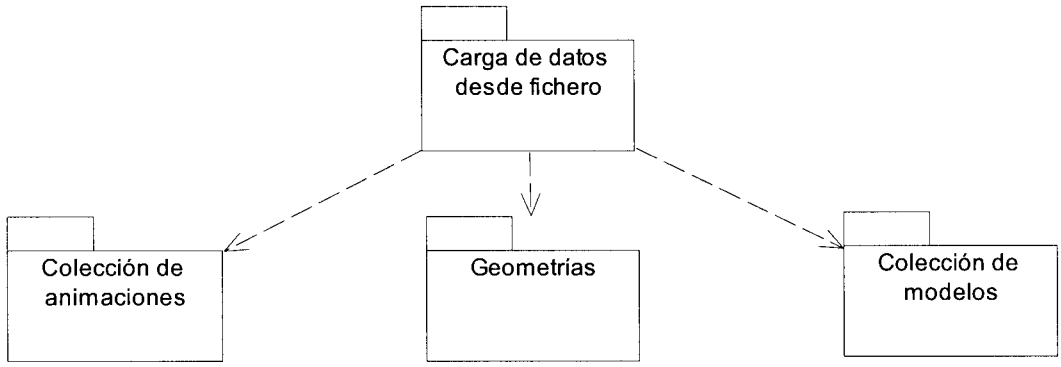

Fig. 20 Diagrama de Paquetes. Paquete "Almacenamiento de datos".

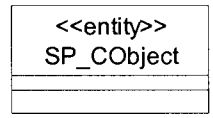

Fig. 21 Diagrama de Clase de Diseño "Objeto".

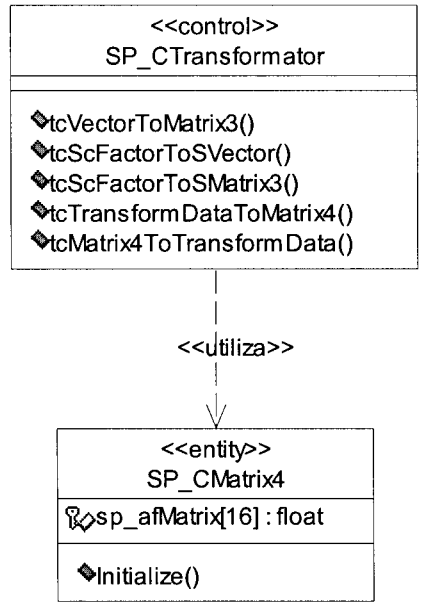

Fig. 22 Diagrama de Clases de Diseño "Transformaciones".

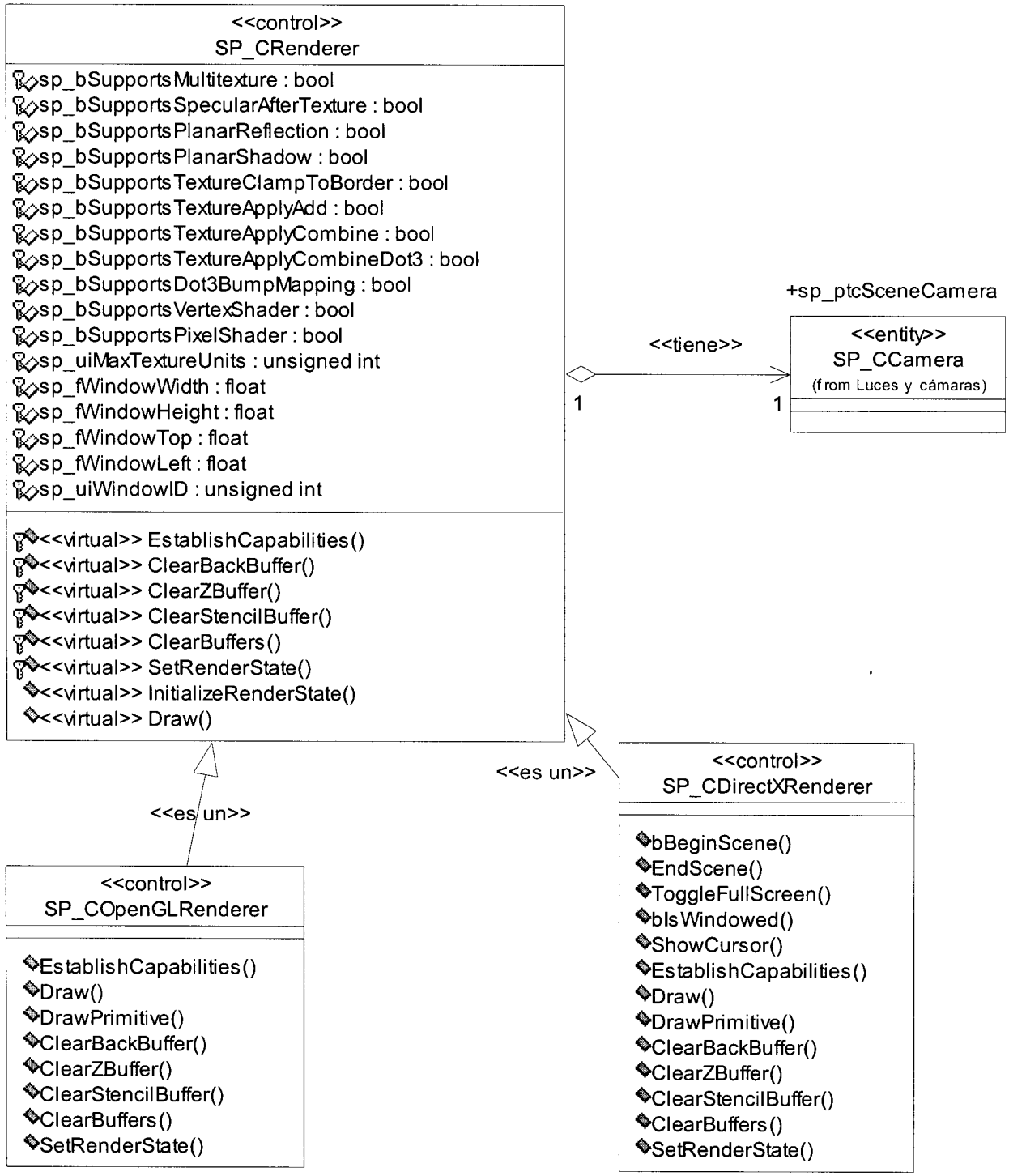

Fig. 23 Diagrama de Clases de Diseño "Render".

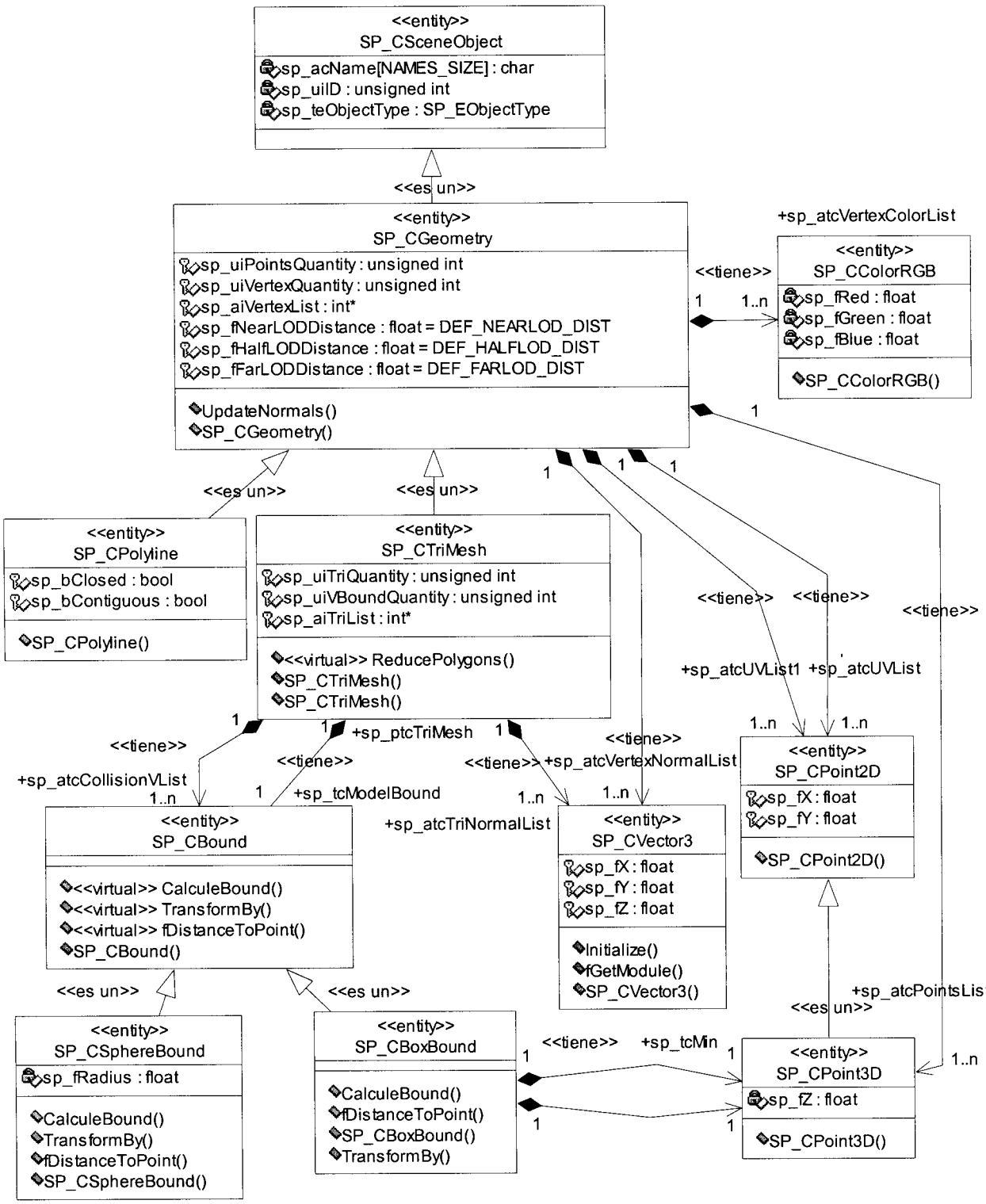

Fig. 24 Diagrama de Clases de Diseño "Geometrías".

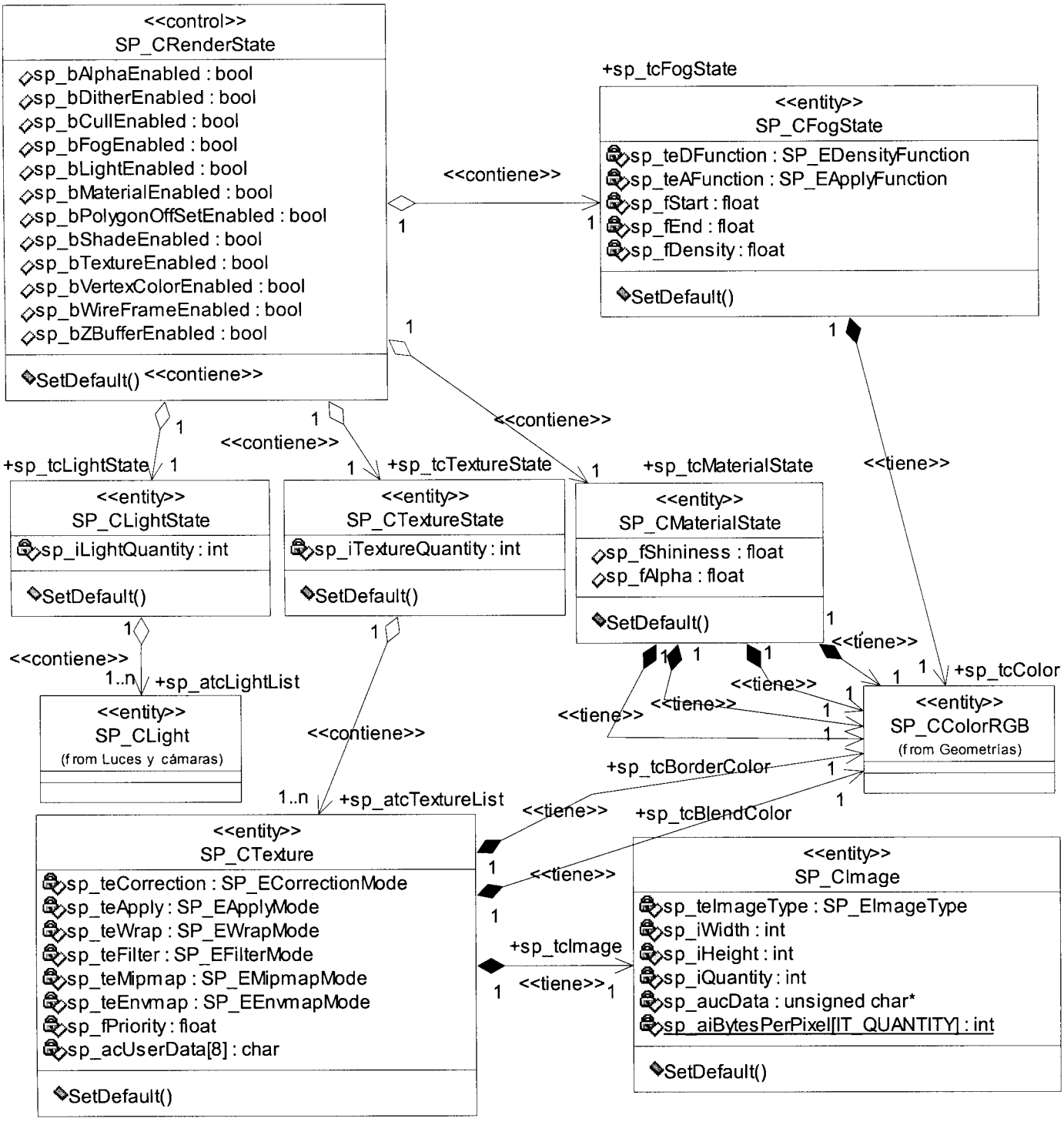

Fig. 25 Diagrama de Clases de Diseño "Estados de render (A)".

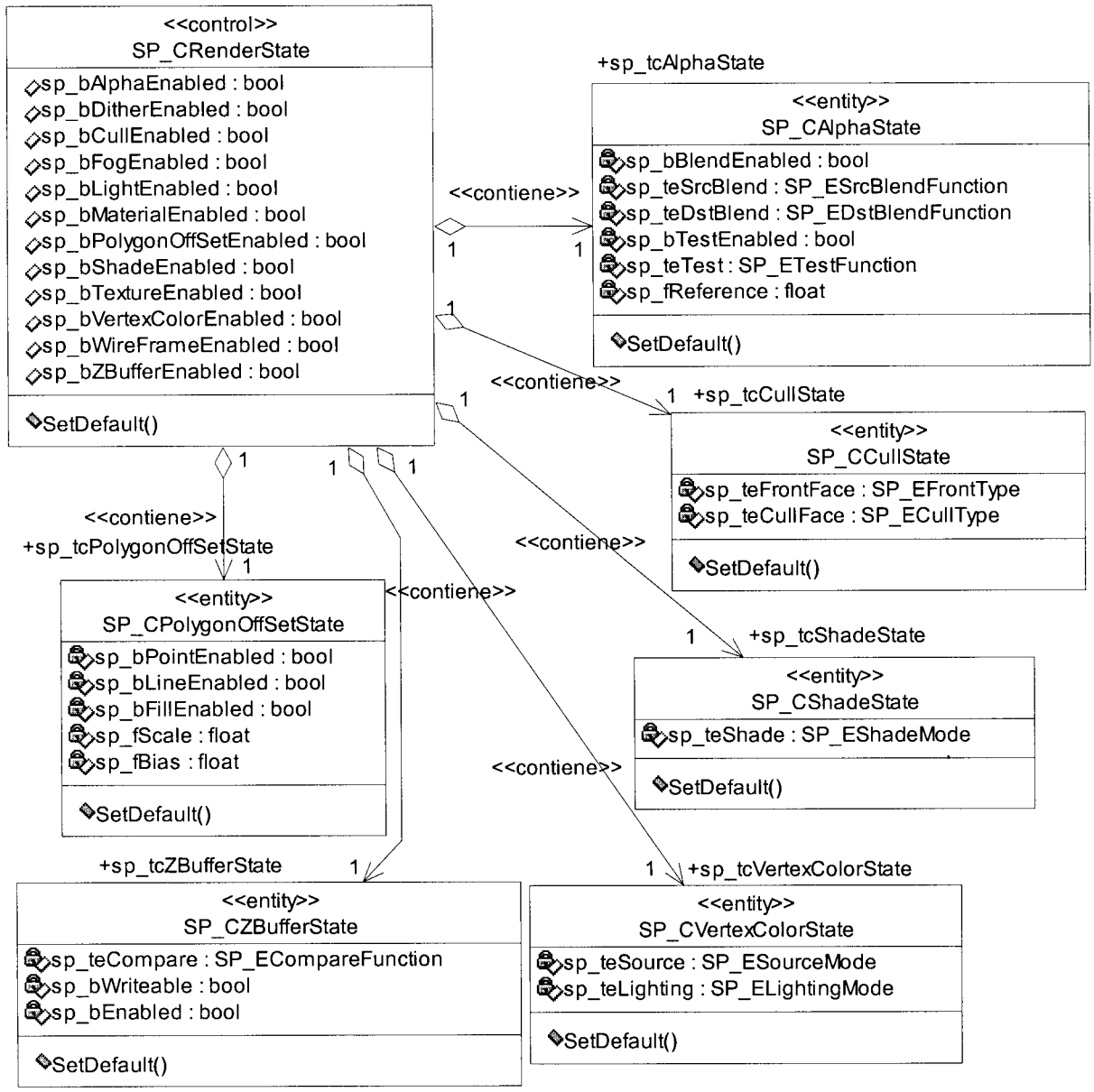

Fig. 26 Diagrama de Clases de Diseño "Estados de render (B)".

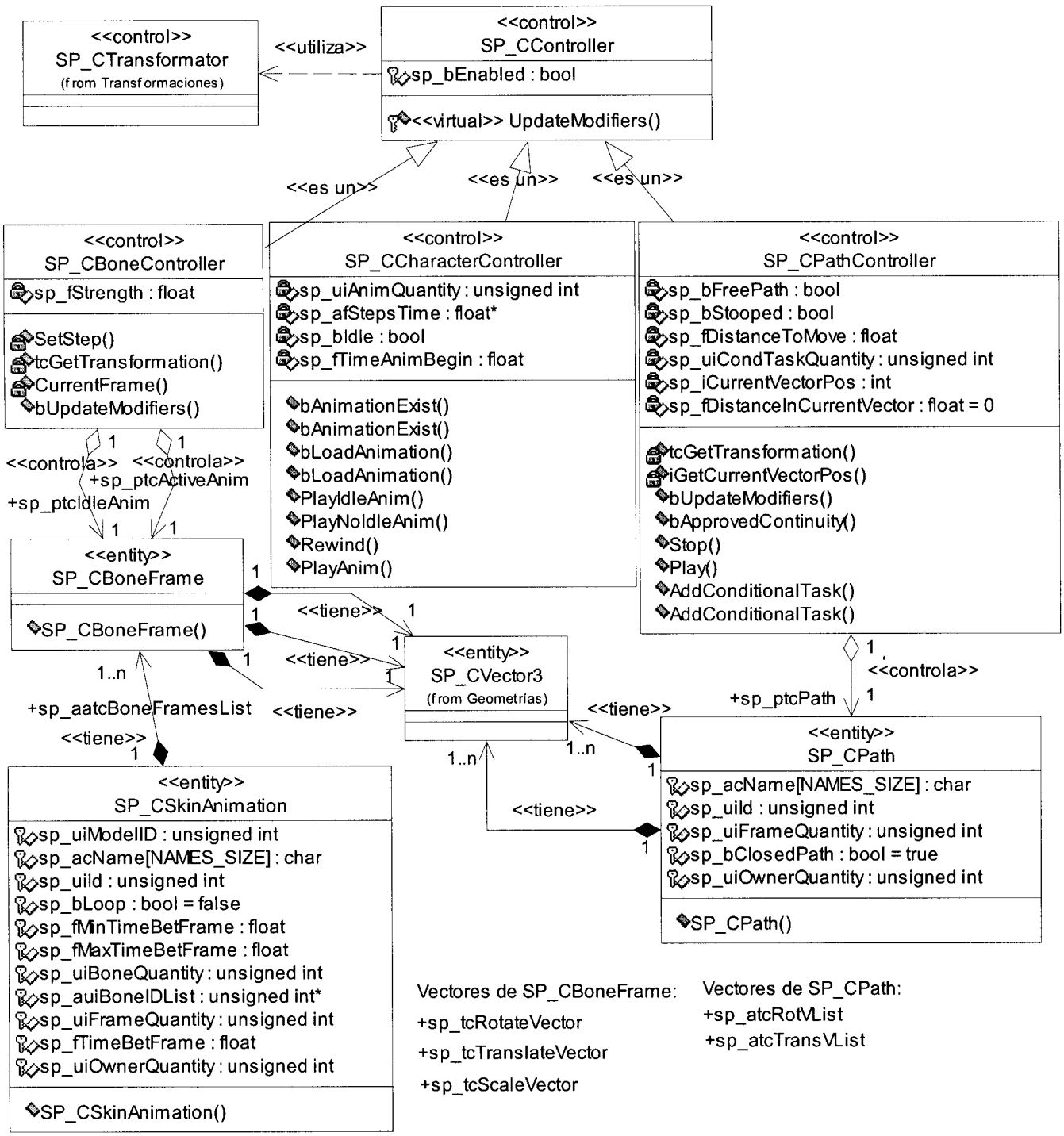

Fig. 27 Diagrama de Clases de Diseño "Manejo de animaciones (A)".

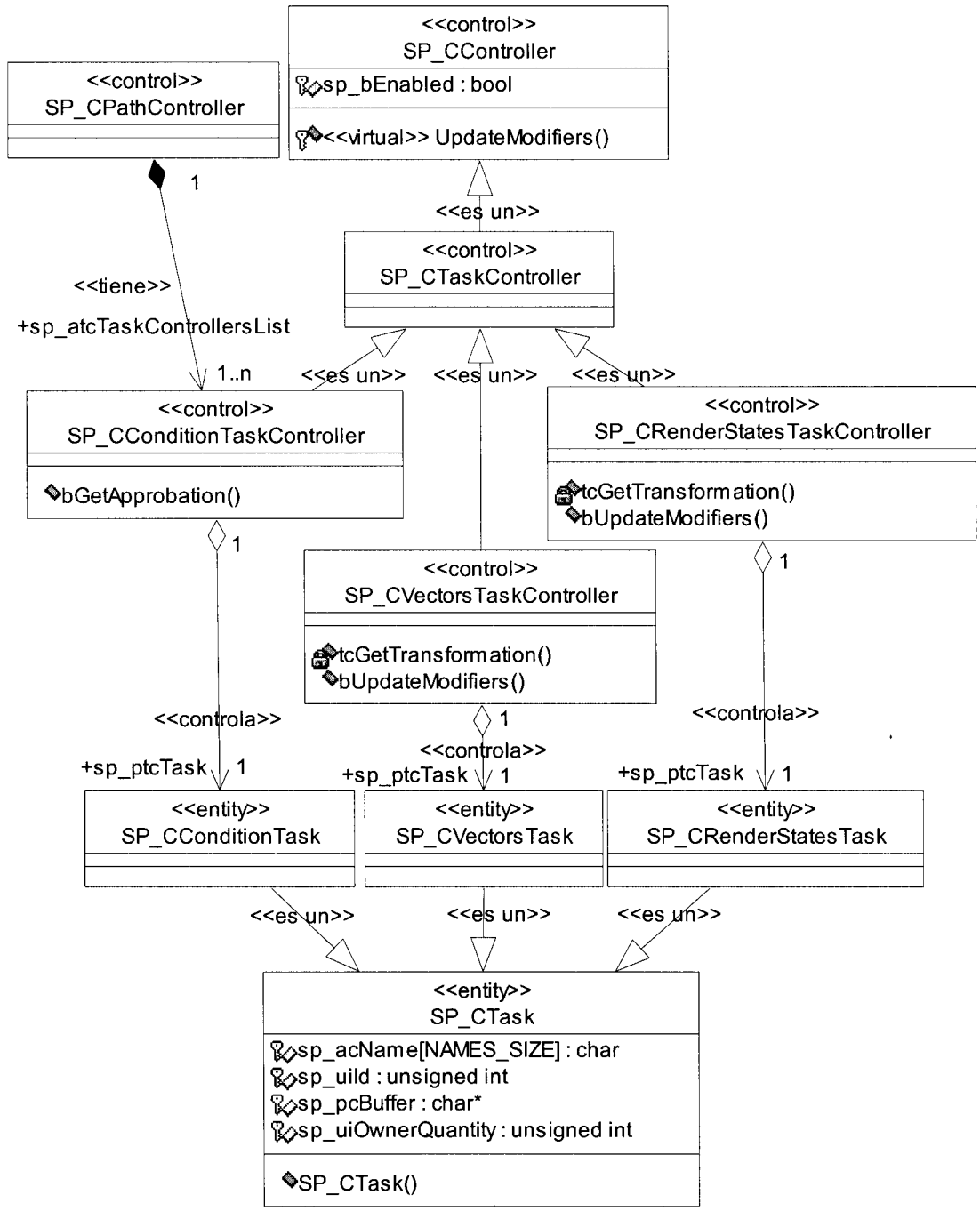

Fig. 28 Diagrama de Clases de Diseño "Manejo de animaciones (B)".

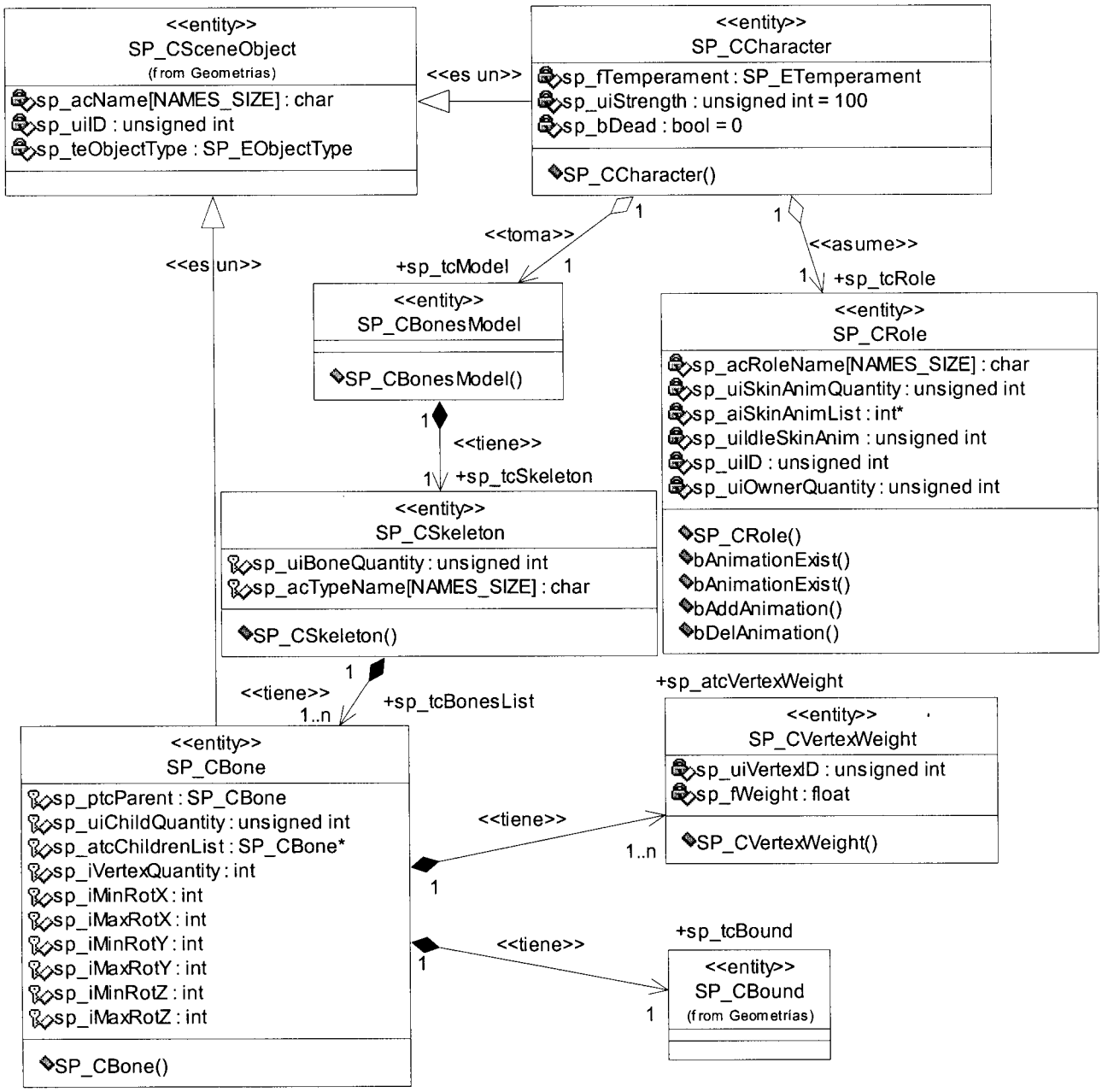

Fig. 29 Diagrama de Clases de Diseño "Personaje".

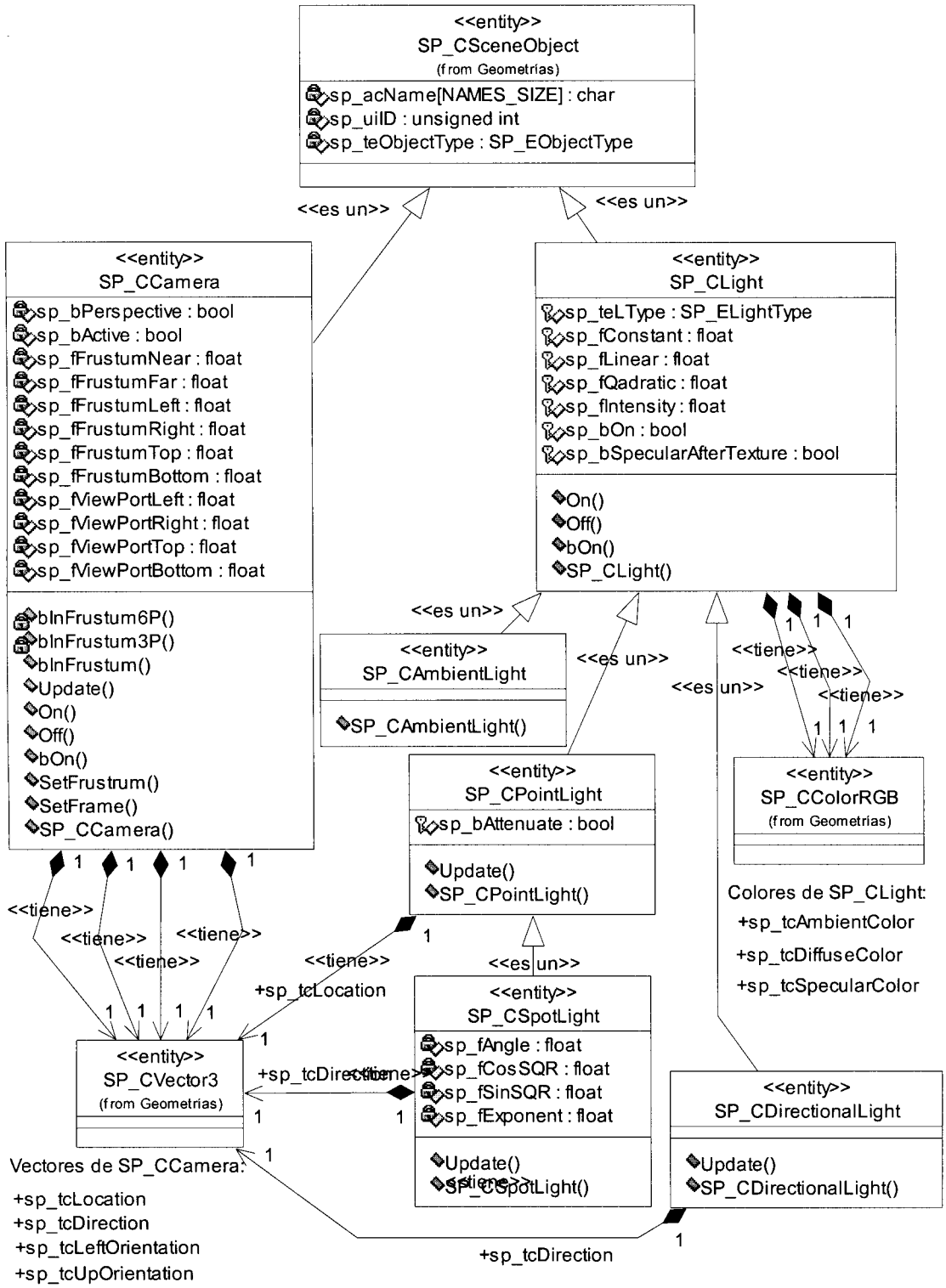

Fig. 30 Diagrama de Clases de Diseño "Luces y cámaras".

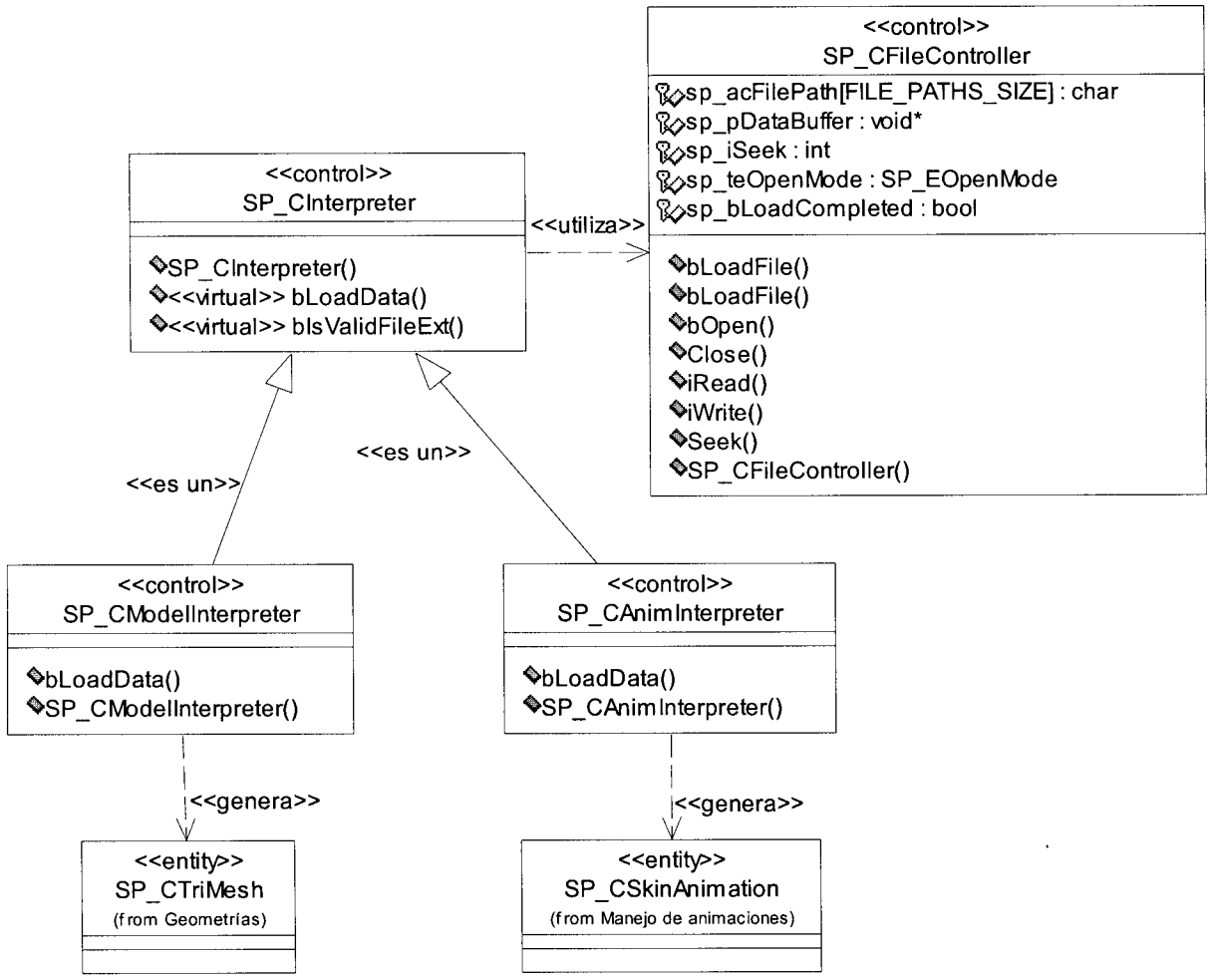

Fig. 31 Diagrama de Clases de Diseño "Carga de datos desde ficheros".

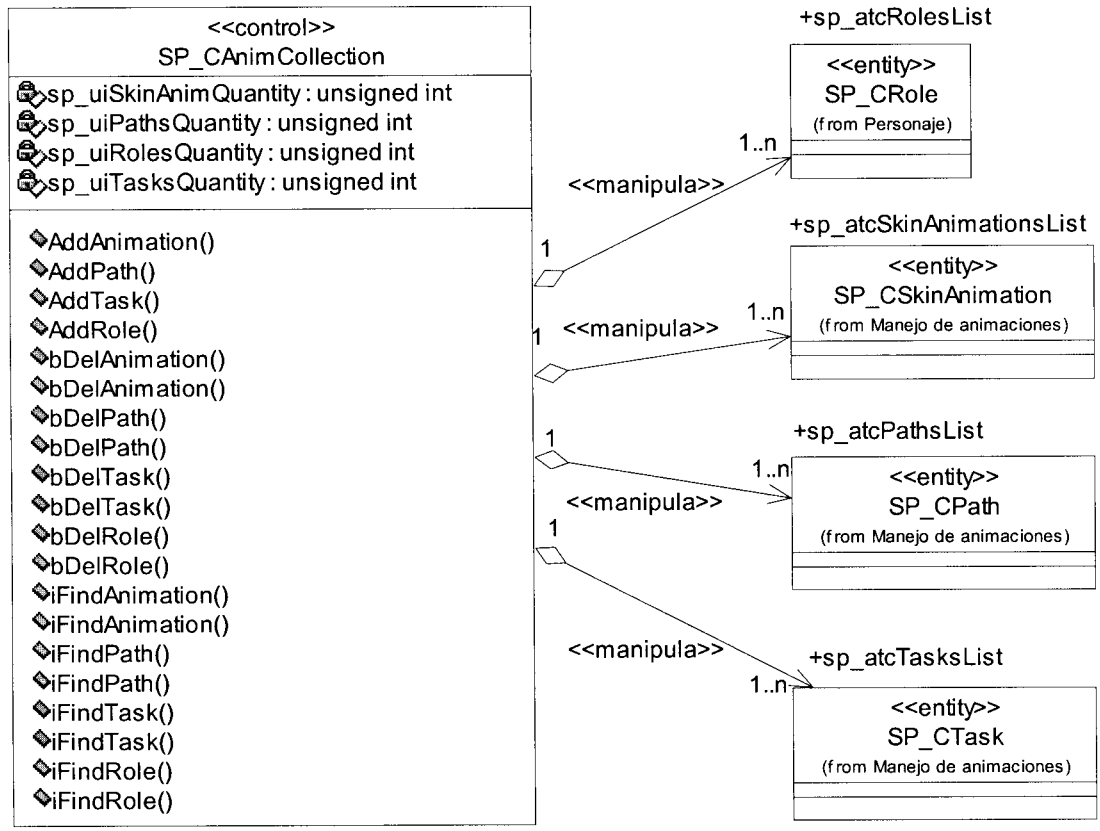

Fig. 32 Diagrama de Clases de Diseño "Colección de animaciones".

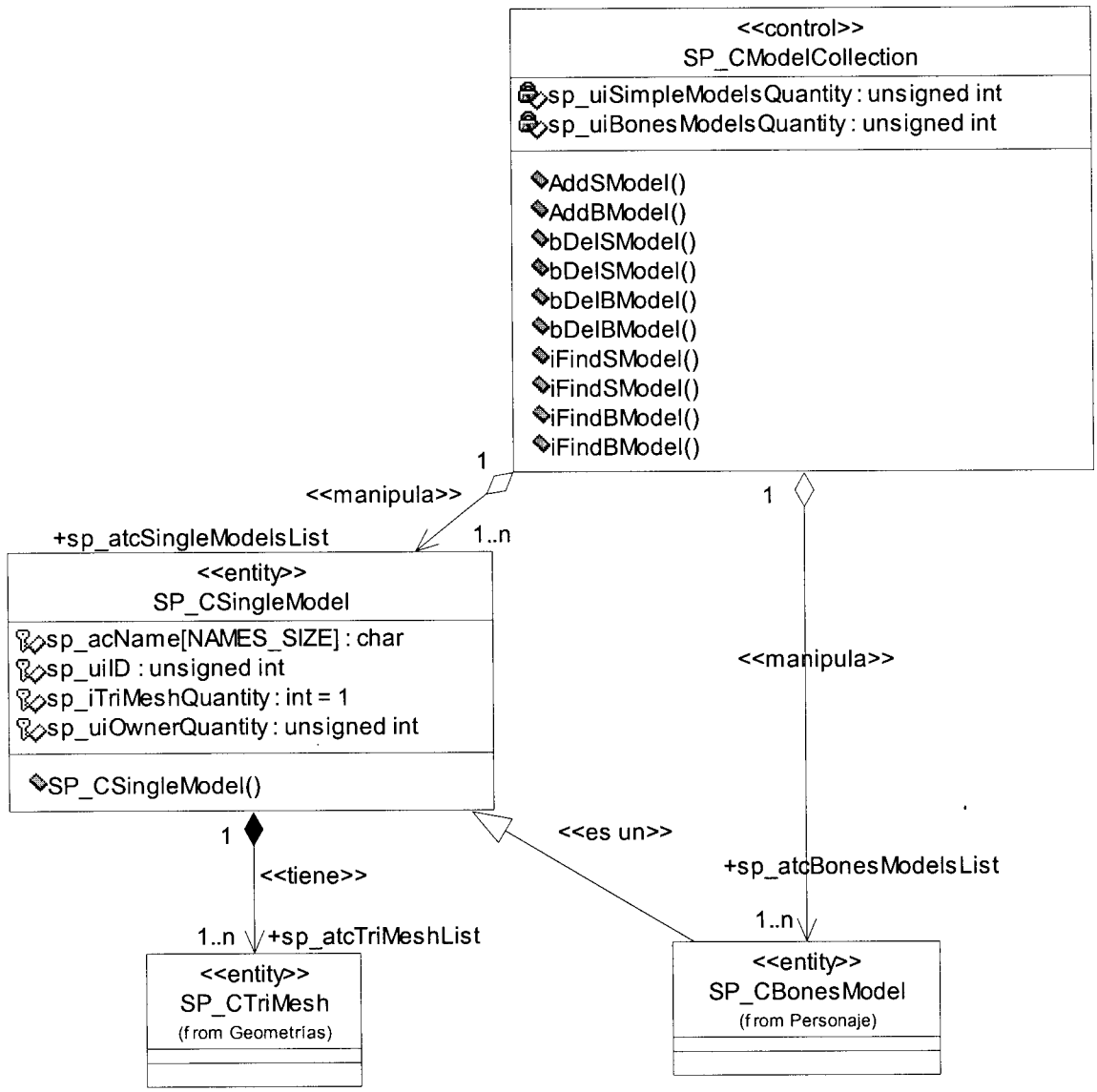

Fig. 33 Diagrama de Clases de Diseño "Colección de modelos".

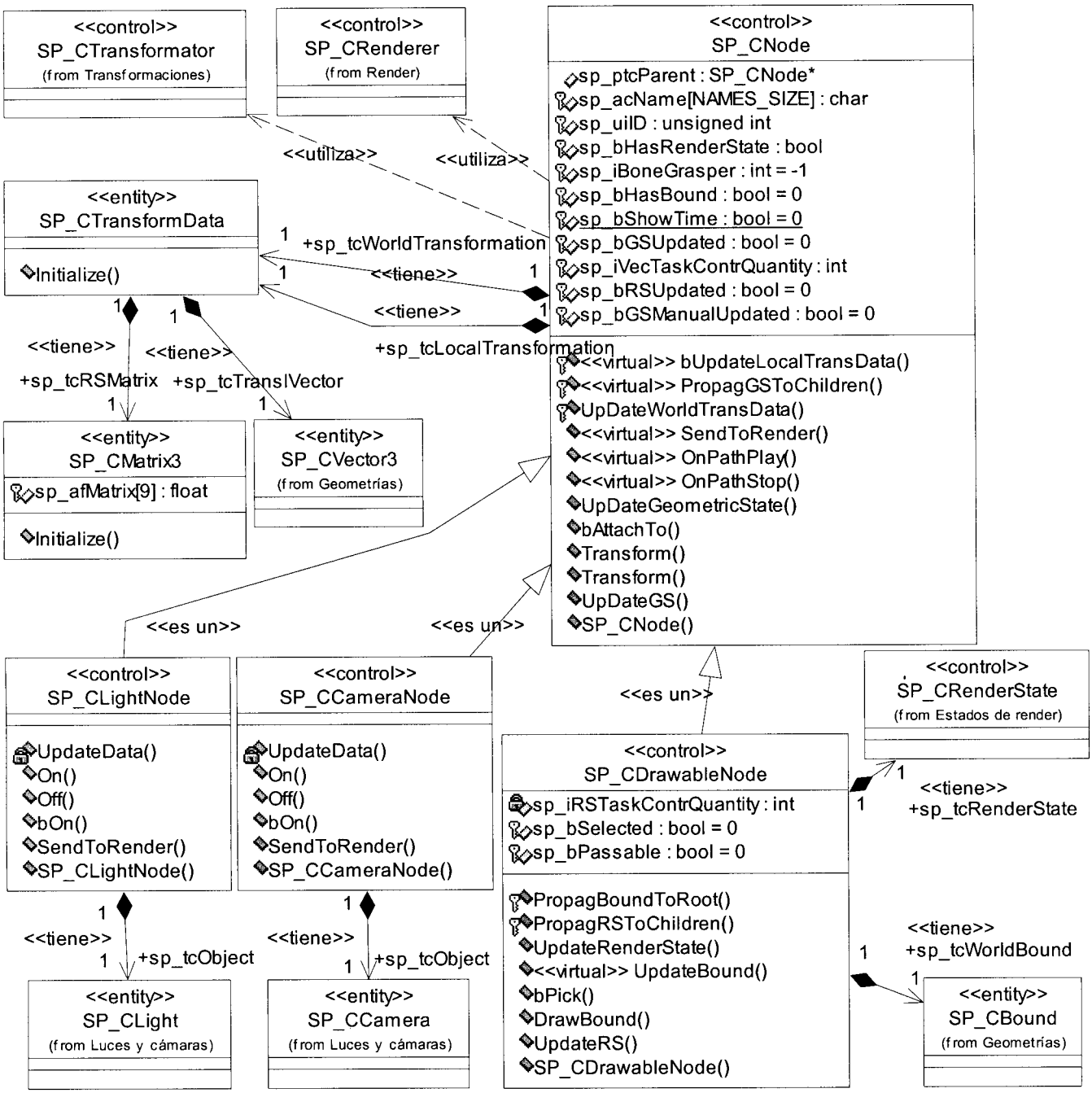

Fig. 34 Diagrama de Clases de Diseño "Manejo del grafo de la escena (A)".

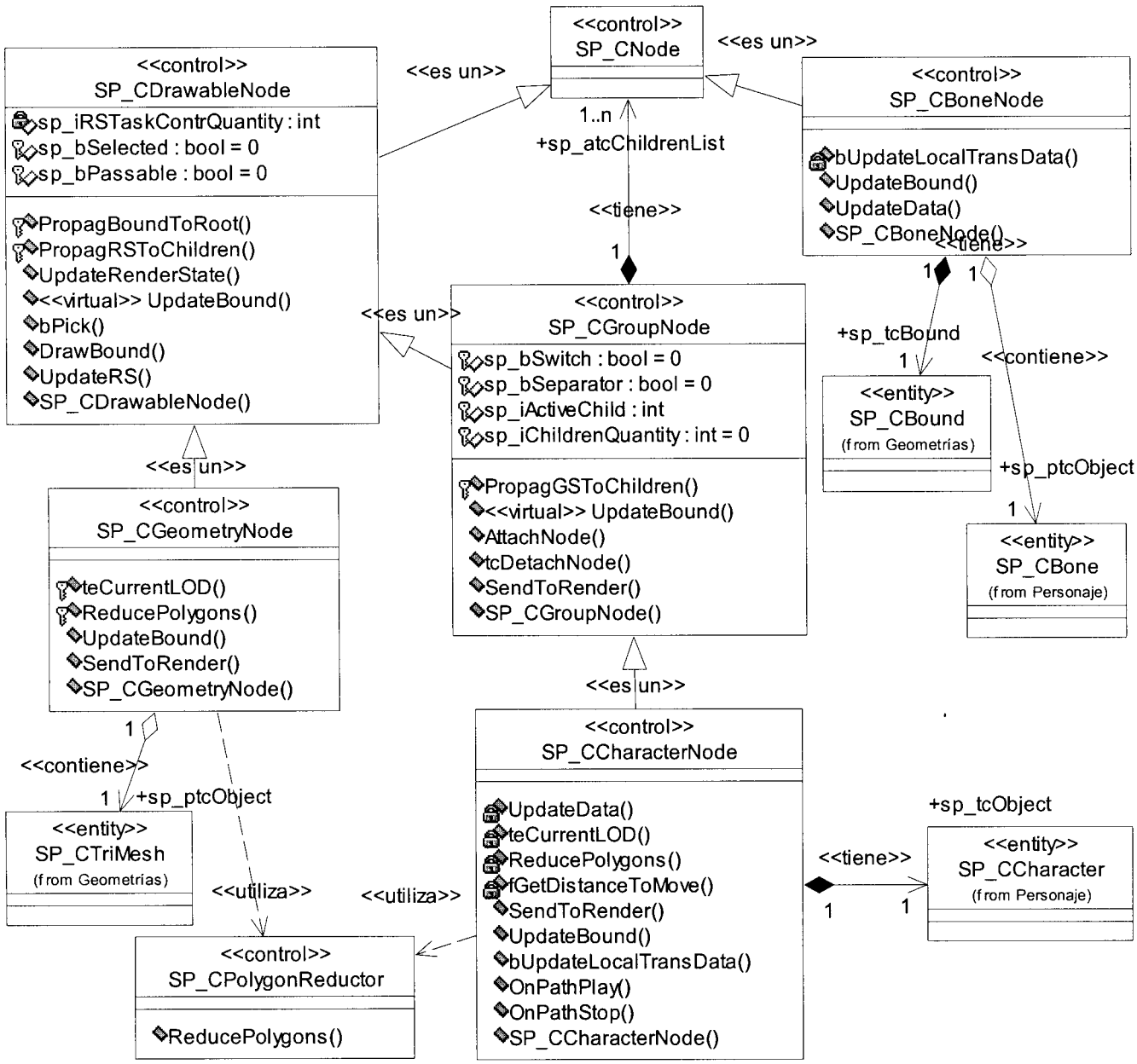

Fig. 35 Diagrama de Clases de Diseño "Manejo del grafo de la escena (B)".

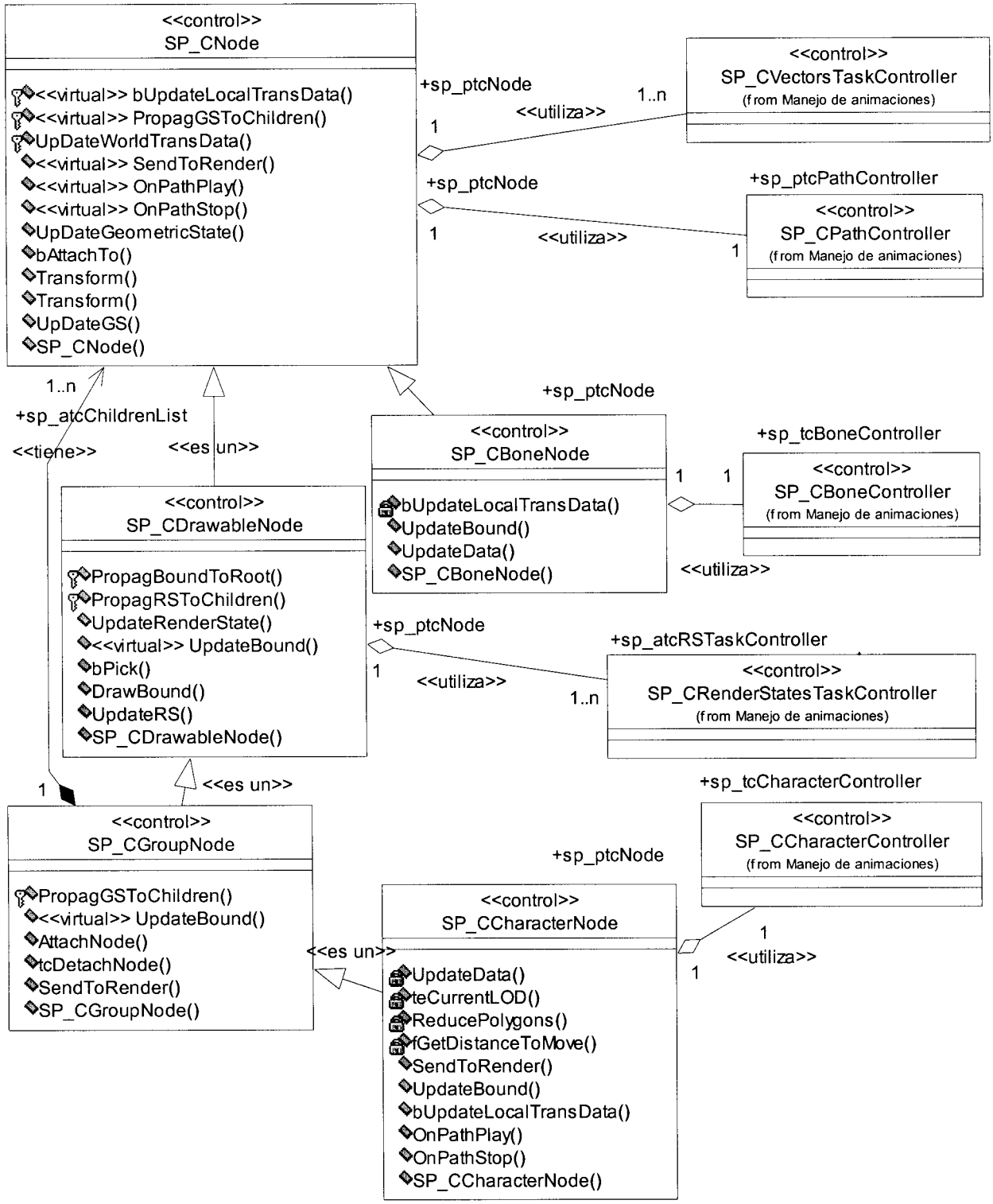

Fig. 36 Diagrama de Clases de Diseño "Manejo del grafo de la escena (C)".

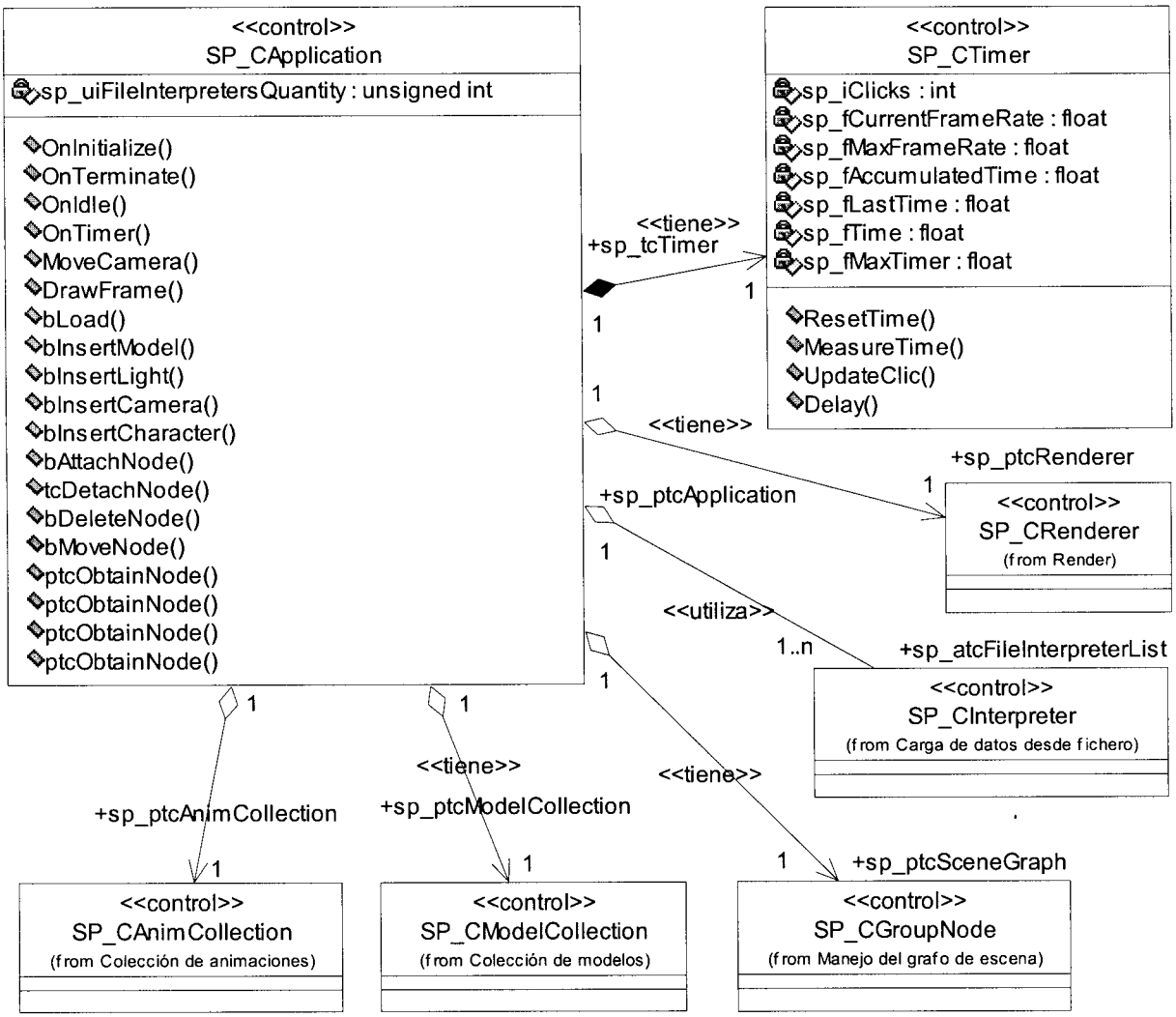

Fig. 37 Diagrama de Clases de Diseño "Aplicación".

4.3 Diagramas de secuencia

A continuación siguen los diagramas de secuencias agrupados paquetes de casos de uso.

4.3.1 Paquete "Almacenar datos"

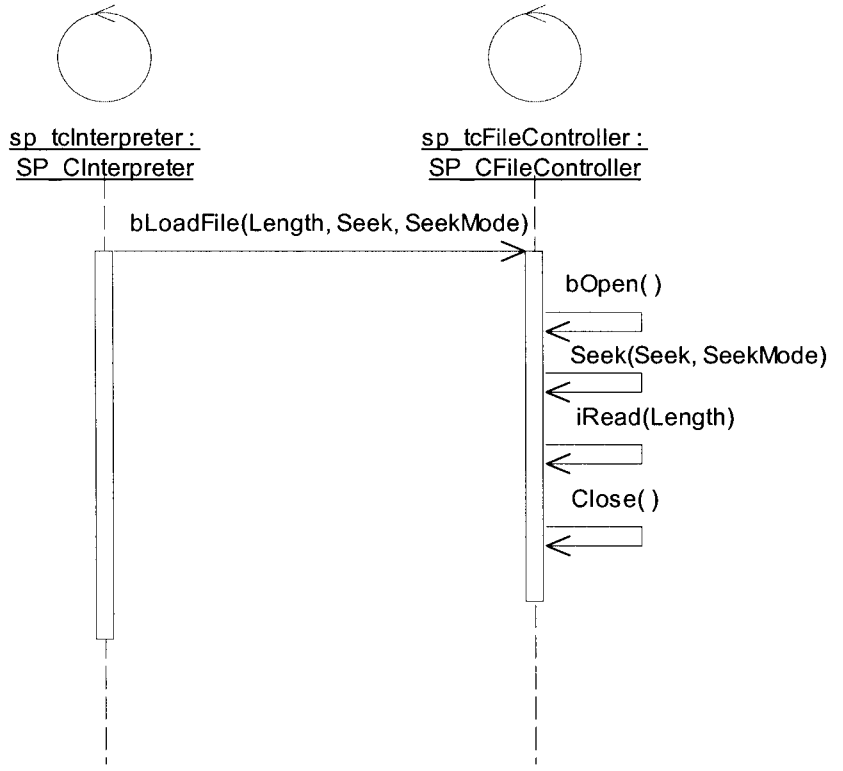

Fig. 38 Diagrama de Secuencia "Cargar ficheros".

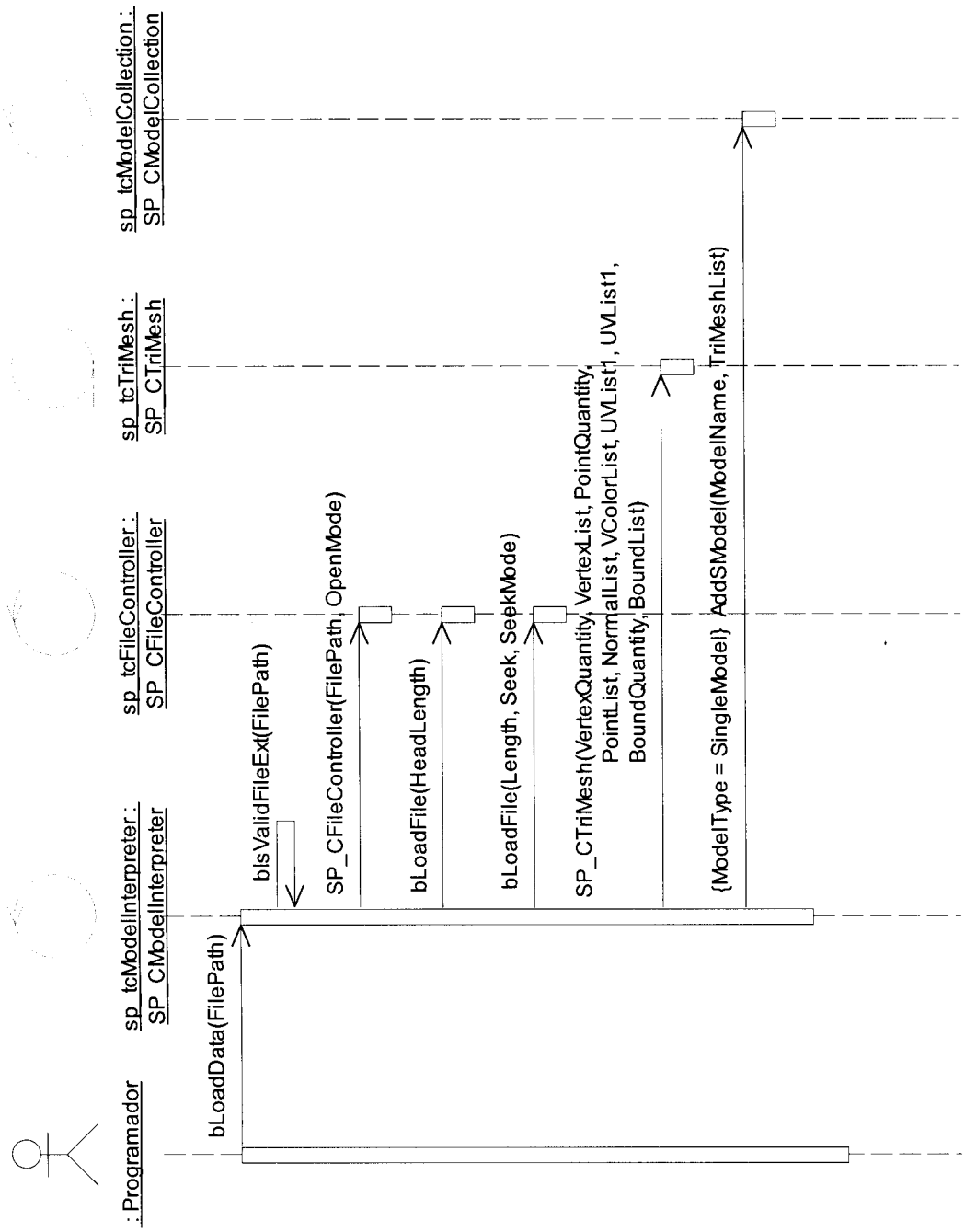

Fig. 39 Diagrama de Secuencia "Cargar modelo simple".

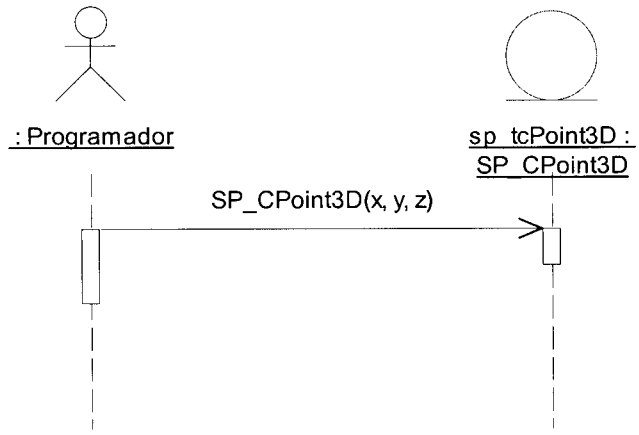

Fig. 40 Diagrama de Secuencia "Crear entidad geométrica. (Sesión Punto)".

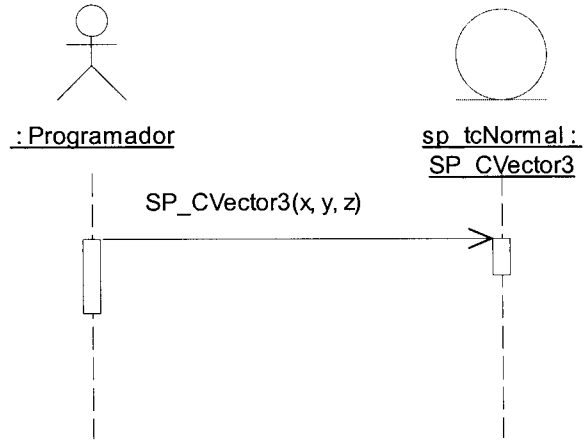

Fig. 41 Diagrama de Secuencia "Crear entidad geométrica. (Sesión Vector)".

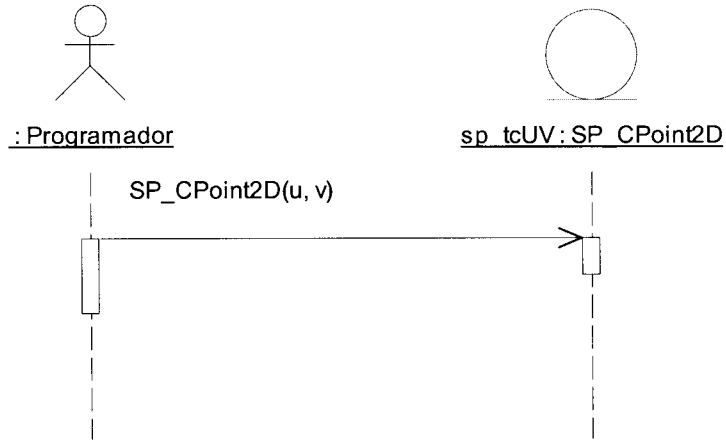

Fig. 42 Diagrama de Secuencia "Crear entidad geométrica. (Sesión Coord. de textura)"

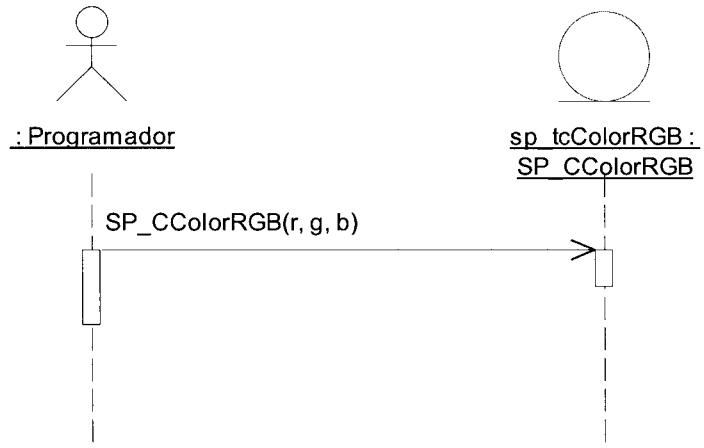

Fig. 43 Diagrama de Secuencia "Crear entidad geométrica. (Sesión Color)".

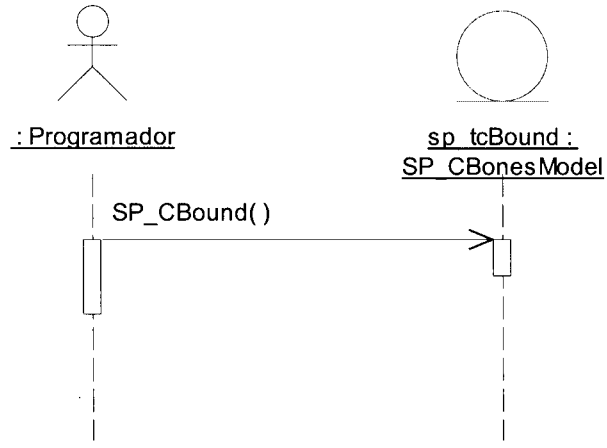

Fig. 44 Diagrama de Secuencia "Crear entidad geométrica. (Sesión Volumen frontera)".

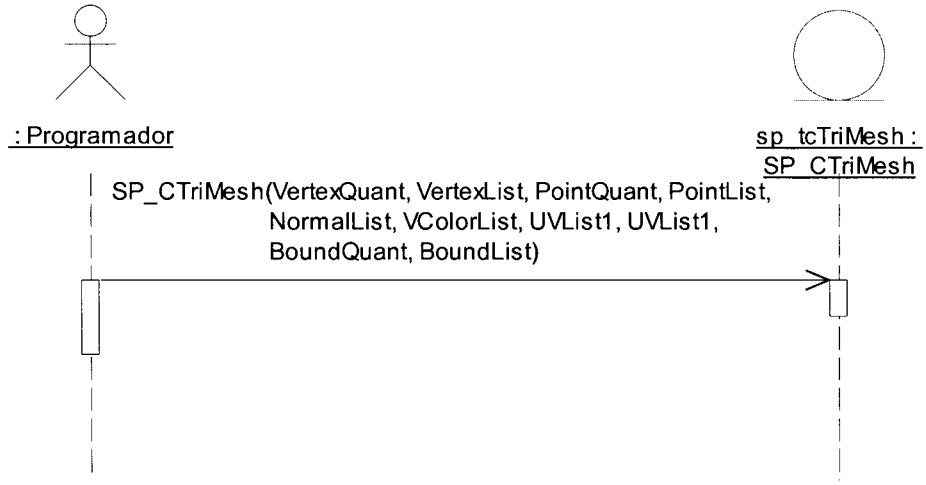

Fig. 45 Diagrama de Secuencia "Crear entidad geométrica. (Sesión Malla)".

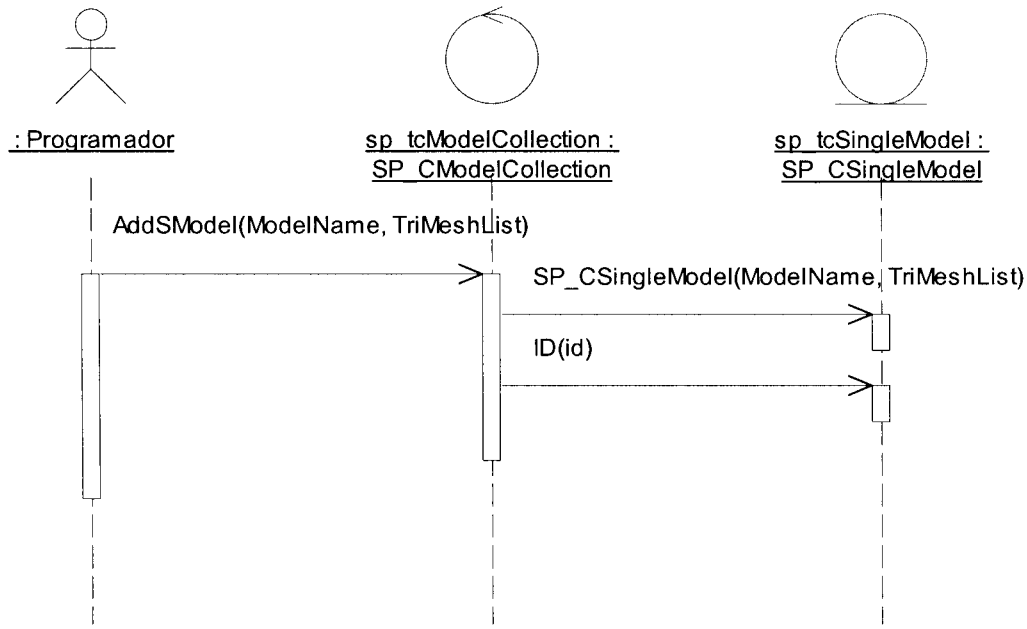

Fig. 46 Diagrama de Secuencia "Adicionar modelo simple a la colección".

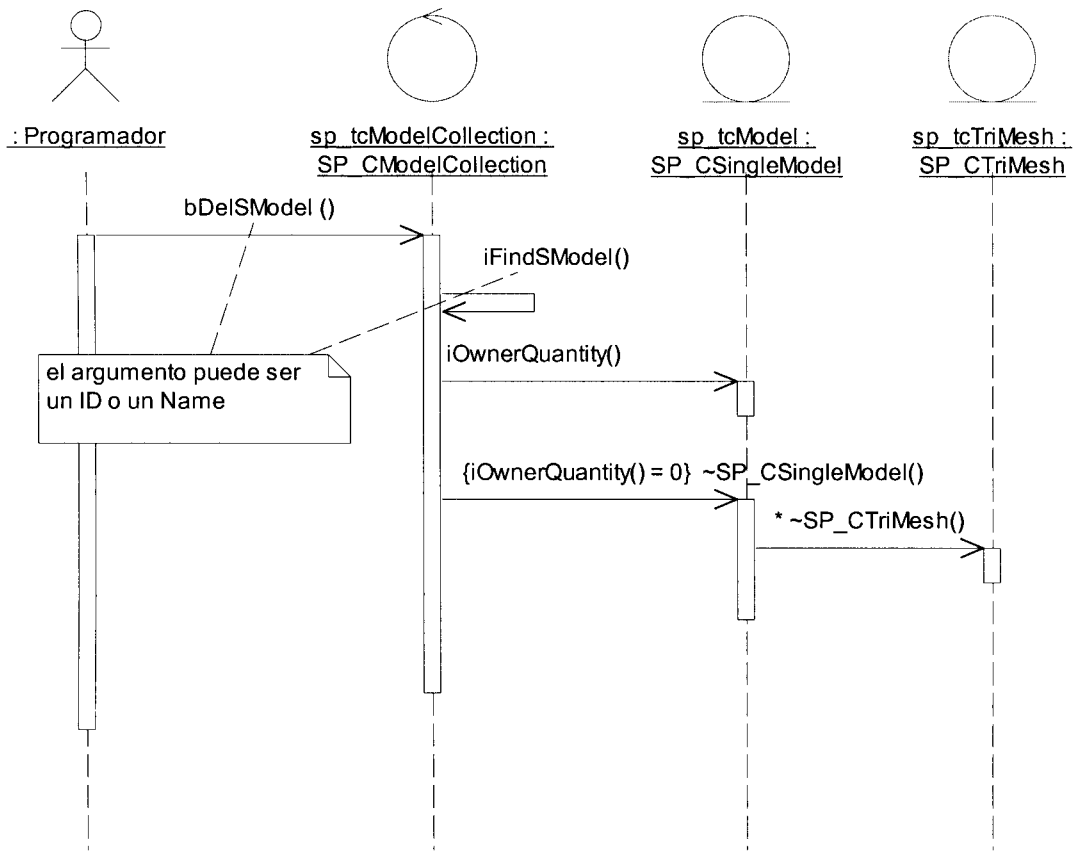

Fig. 47 Diagrama de Secuencia "Eliminar modelo simple de la colección".

4.3.2 Paquete "Manejar grafo de la escena"

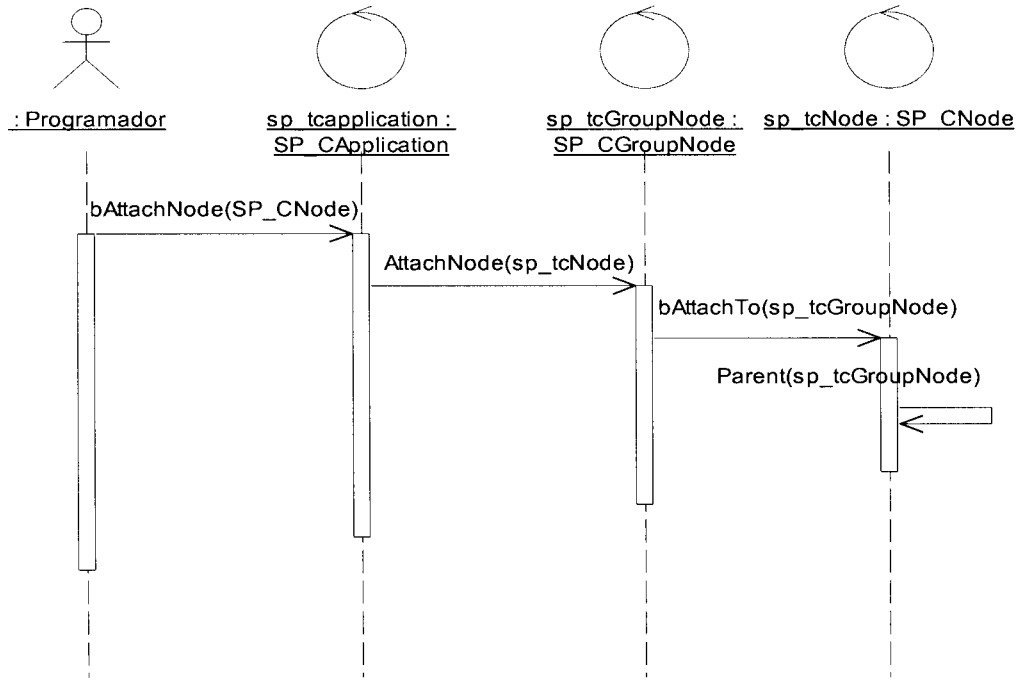

Fig. 48 Diagrama de Secuencia "Insertar nodo".

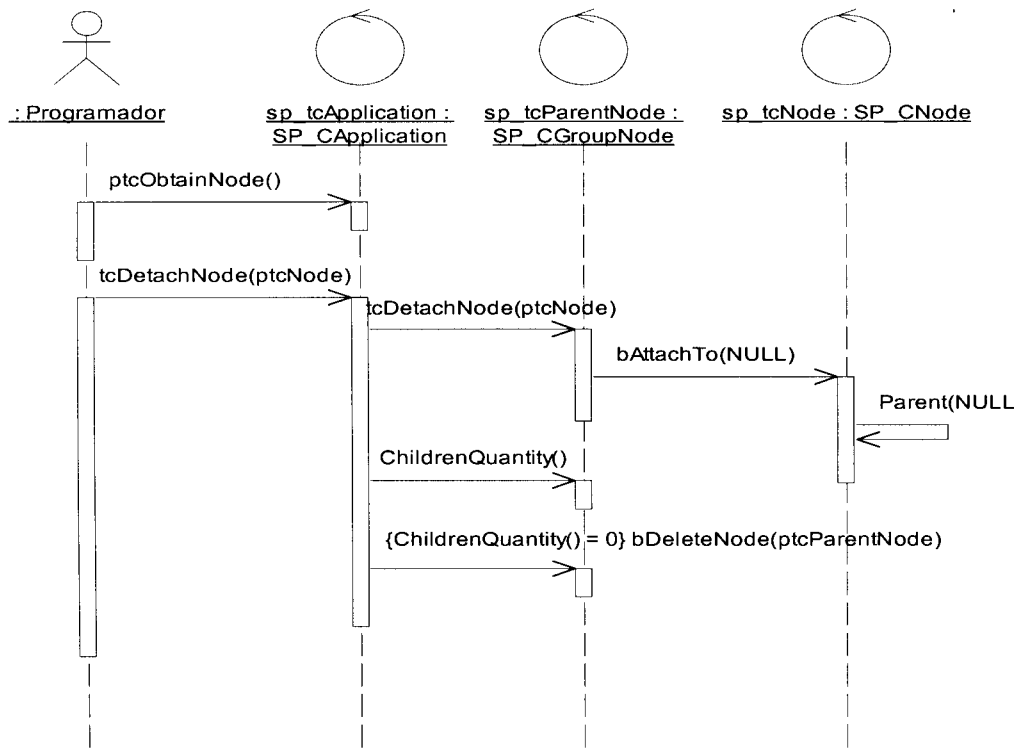

Fig. 49 Diagrama de Secuencia "Eliminar nodo".

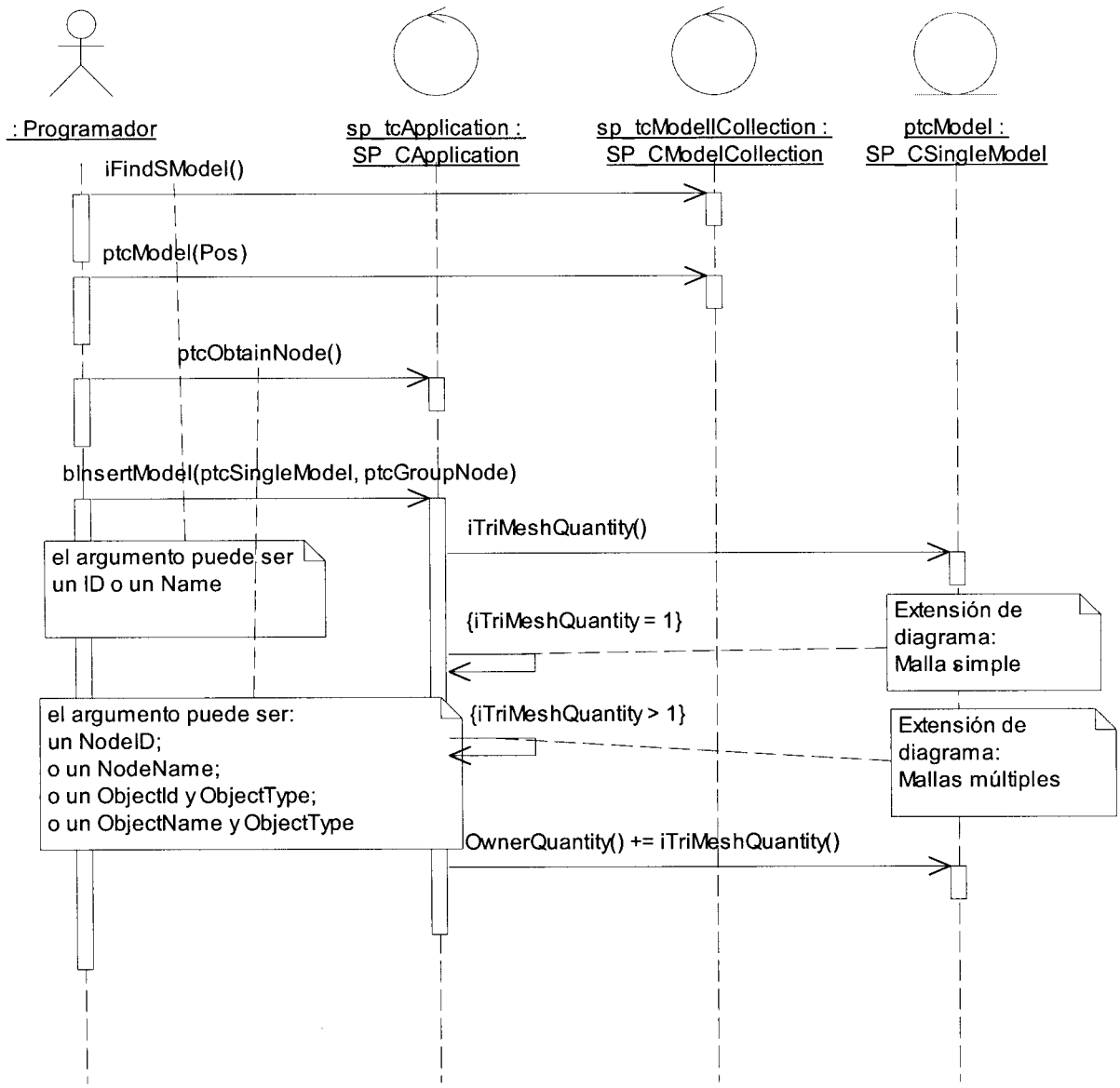

Fig. 50 Diagrama de Secuencia "Crear instancia de modelo simple en escena".

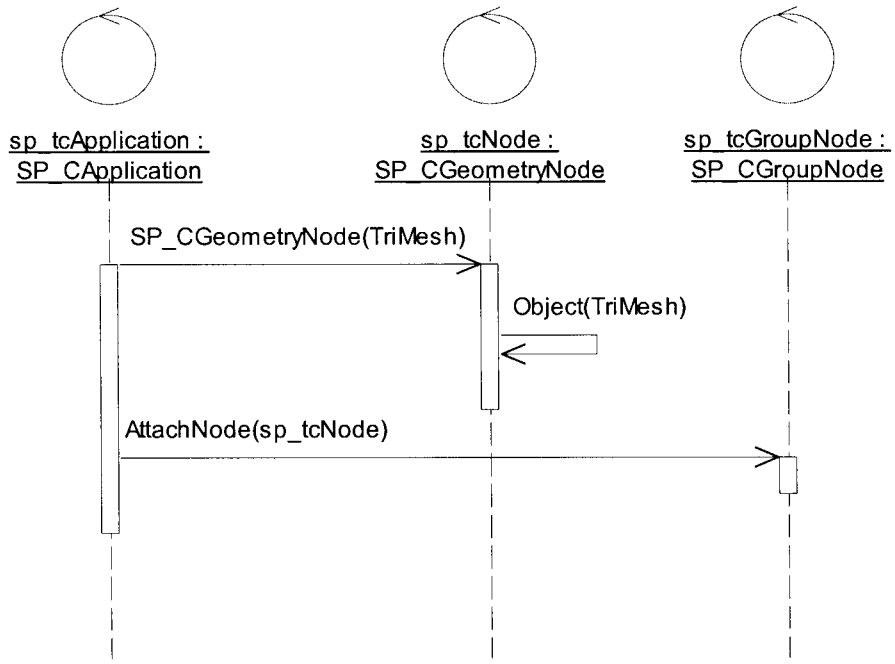

Fig. 51 Diagrama de Secuencia "Crear inst. de mod. simple en escena (Ext. Malla simple)"

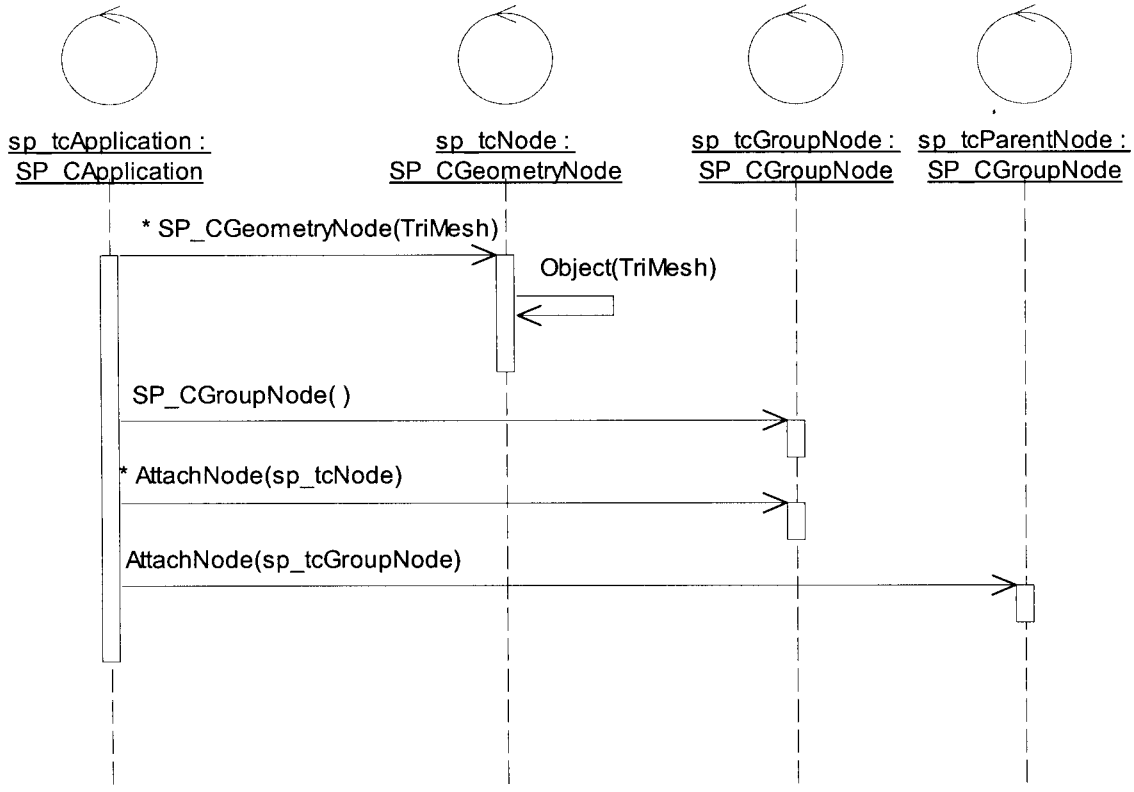

Fig. 52 Diagrama de Secuencia "Crear inst. de mod. simple en escena (Ext. Malla múltiple)"

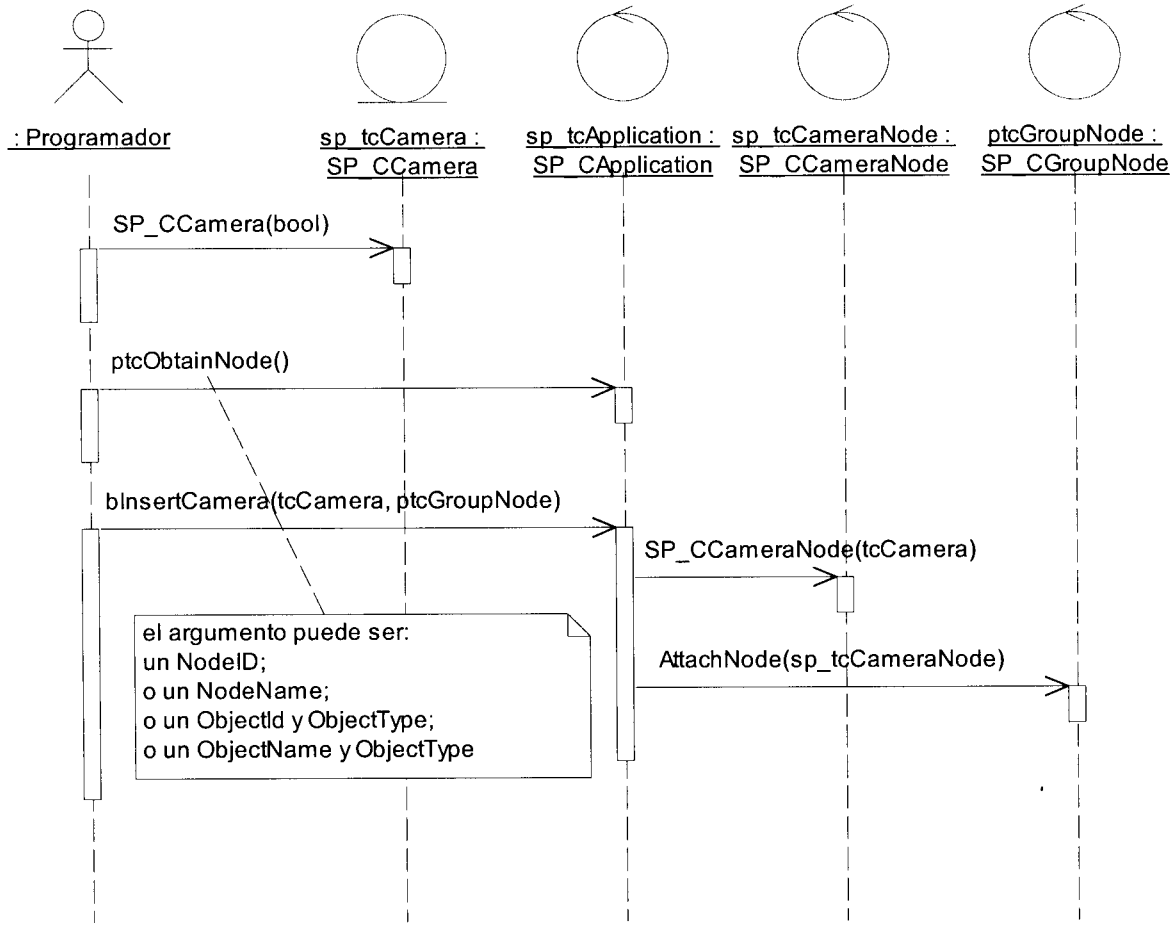

Fig. 53 Diagrama de Secuencia "Crear luz o cámara (Sesión Cámara)".

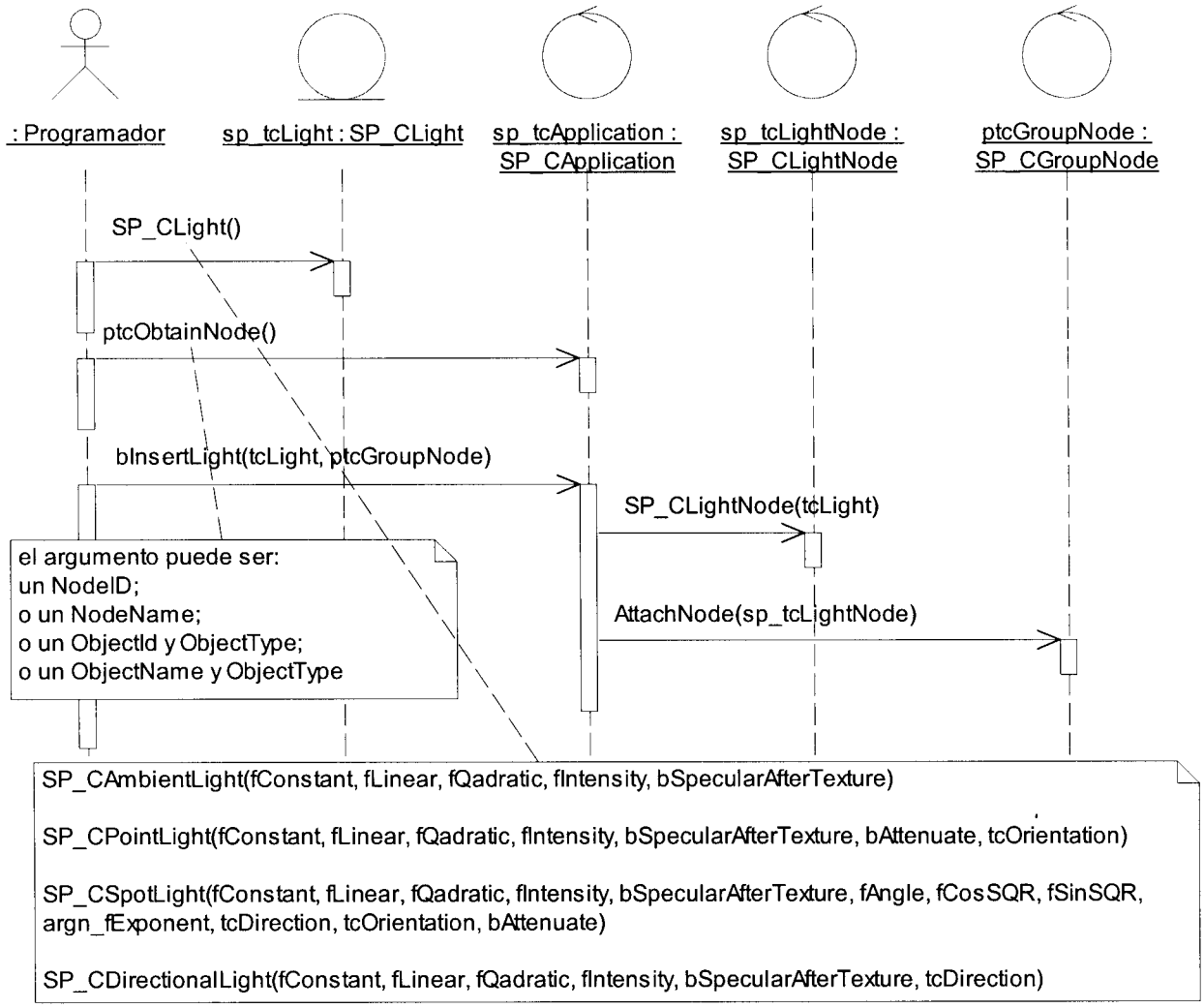

Fig. 54 Diagrama de Secuencia "Crear luz o cámara (Sesión Luz)".

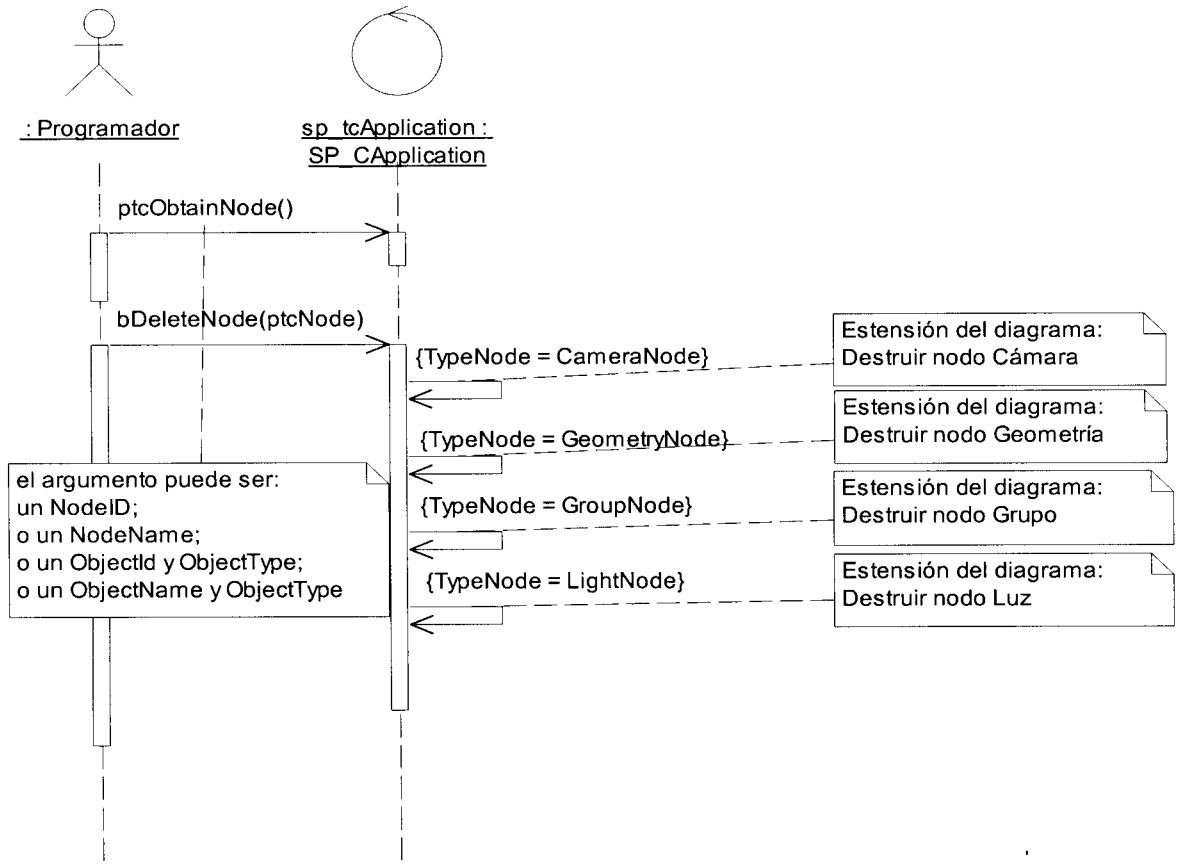

Fig. 55 Diagrama de Secuencia "Destruir nodo".

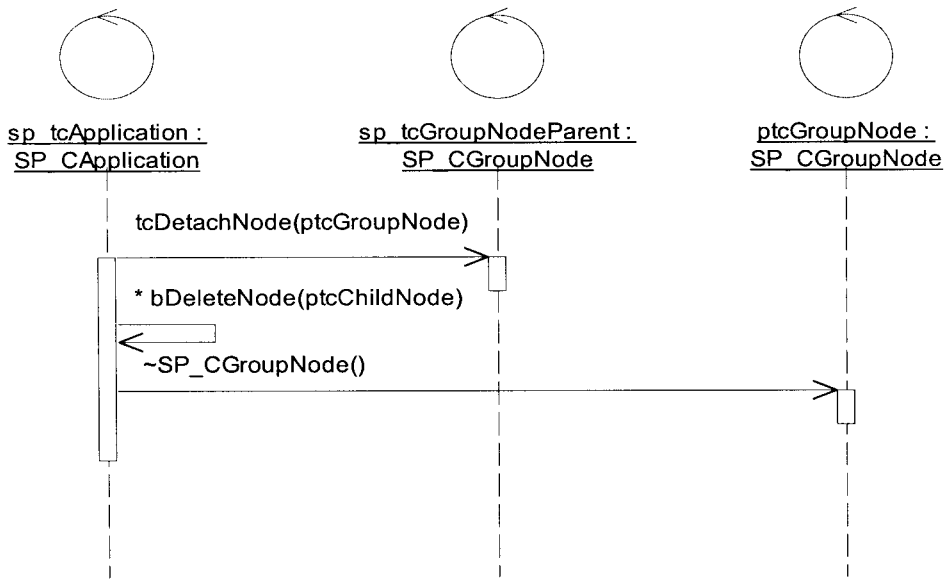

Fig. 56 Diagrama de Secuencia "Destruir nodo (Extensión Grupo)".

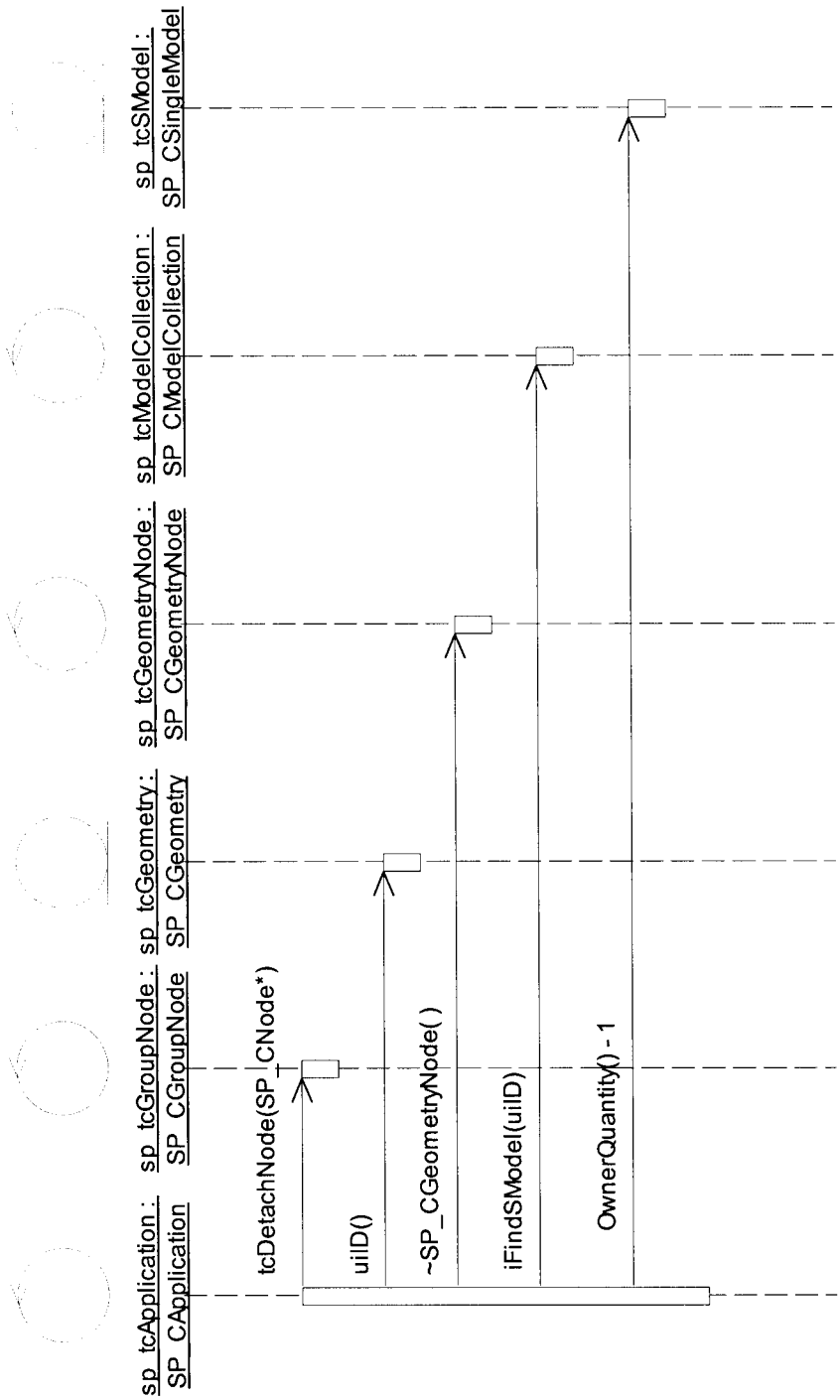

Fig. 57 Diagrama de Secuencia "Destruir nodo (Extensión Geometría)".

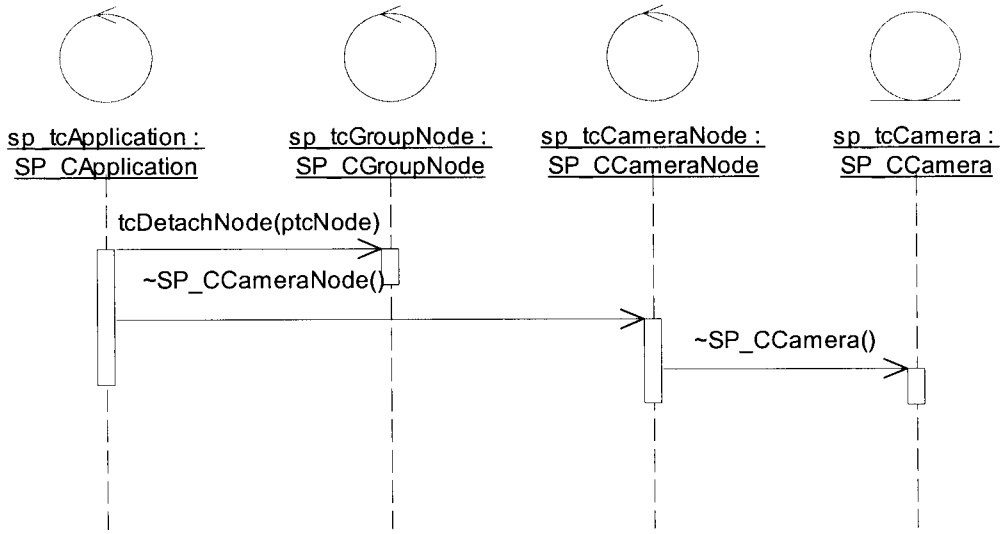

Fig. 58 Diagrama de Secuencia "Destruir nodo (Extensión Cámara)".

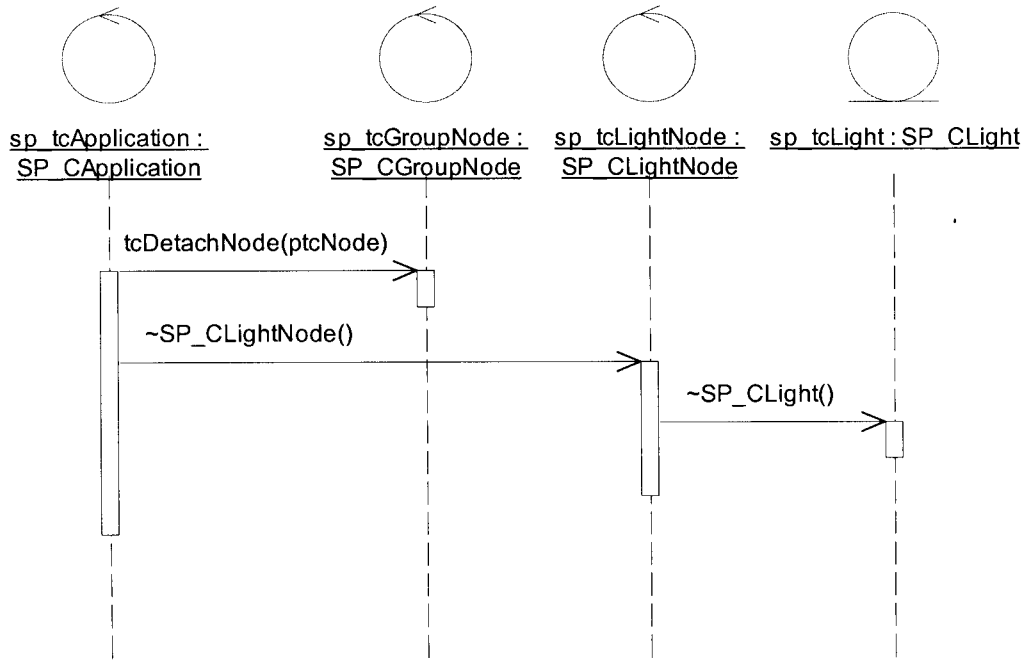

Fig. 59 Diagrama de Secuencia "Destruir nodo (Extensión Luz)".

4.3.3 Paquete "Modificar manualmente"

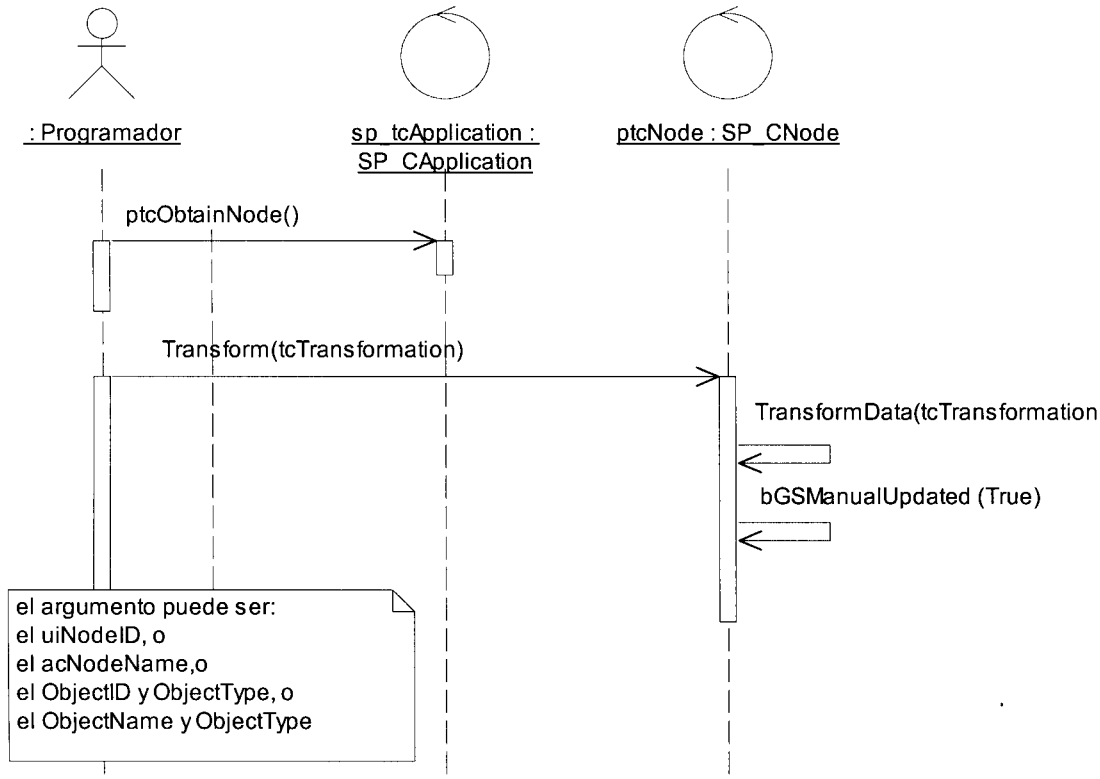

Fig. 60 Diagrama de Secuencia "Actualizar EG manualmente (Sesión TransformData)".

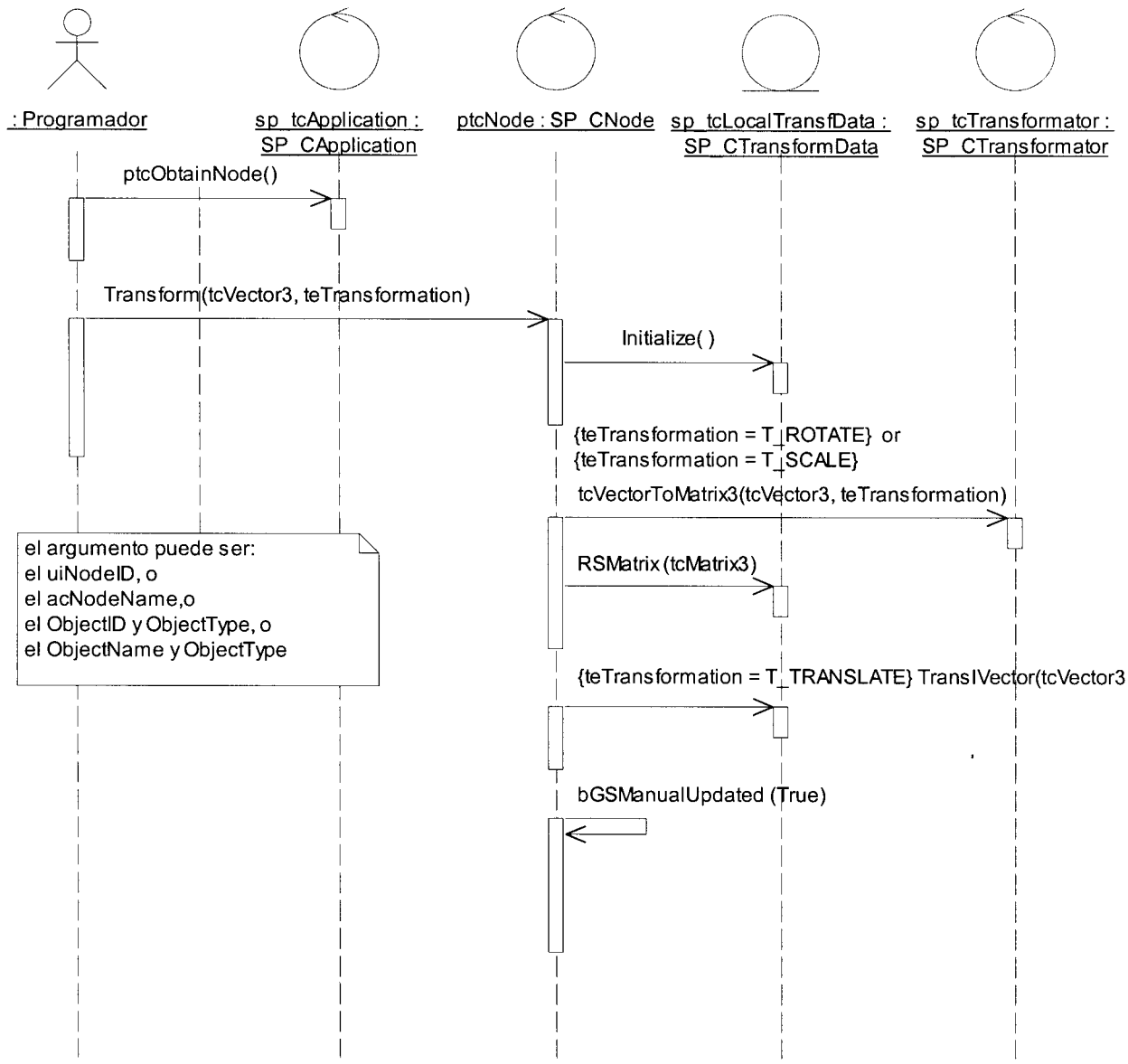

Fig. 61 Diagrama de Secuencia "Actualizar EG manualmente (Sesión Vectores)".

4.3.4 Paquete "Hacer ciclo"

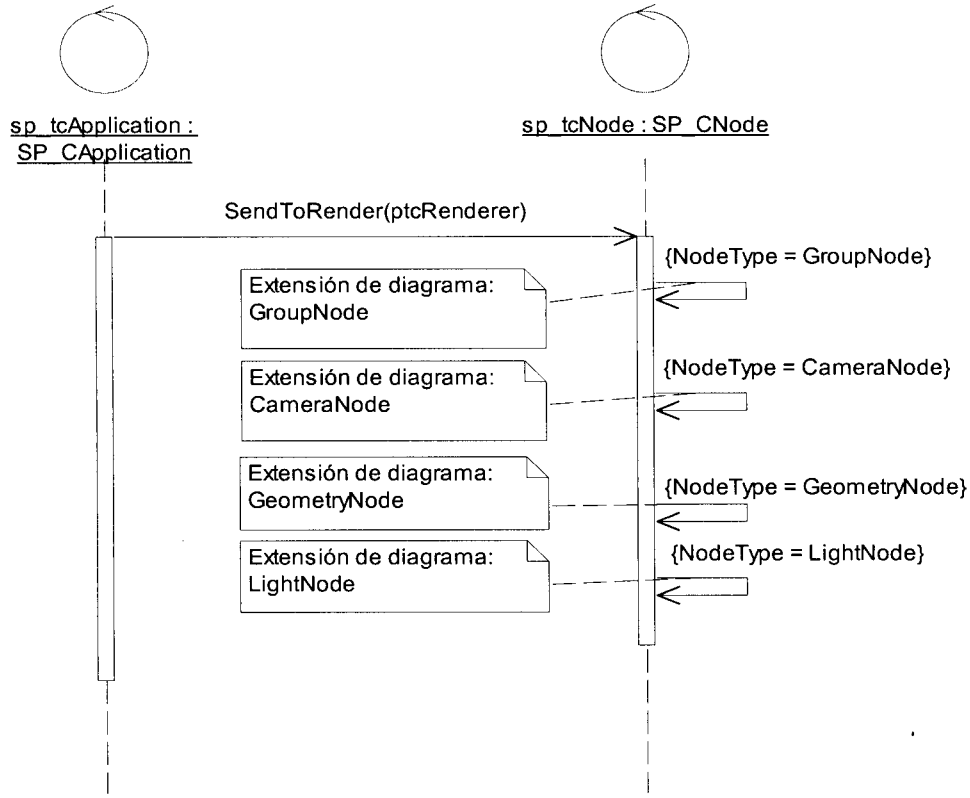

Fig. 62 Diagrama de Secuencia "Enviar a render".

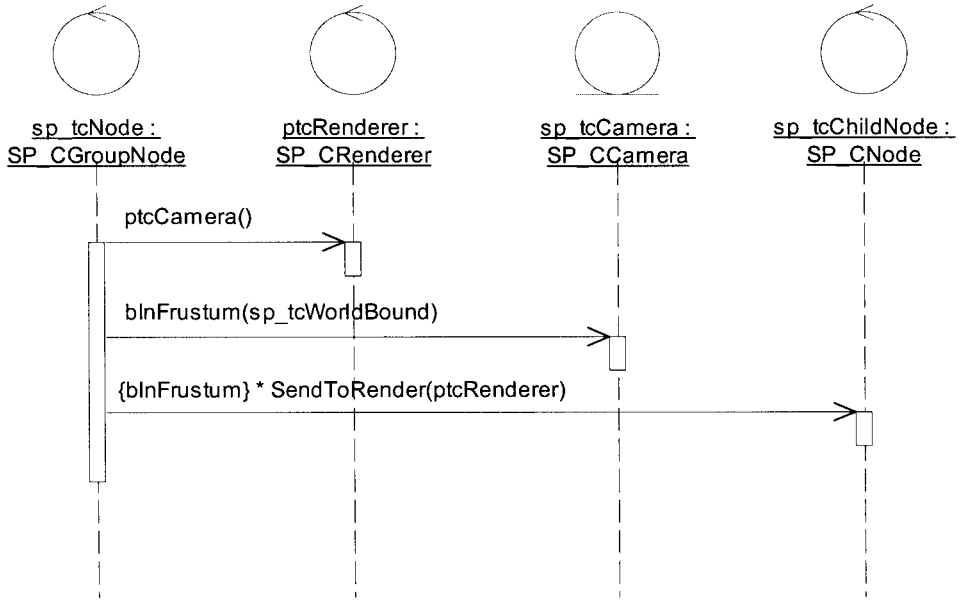

Fig. 63 Diagrama de Secuencia "Enviar a render (Extensión Grupo)".

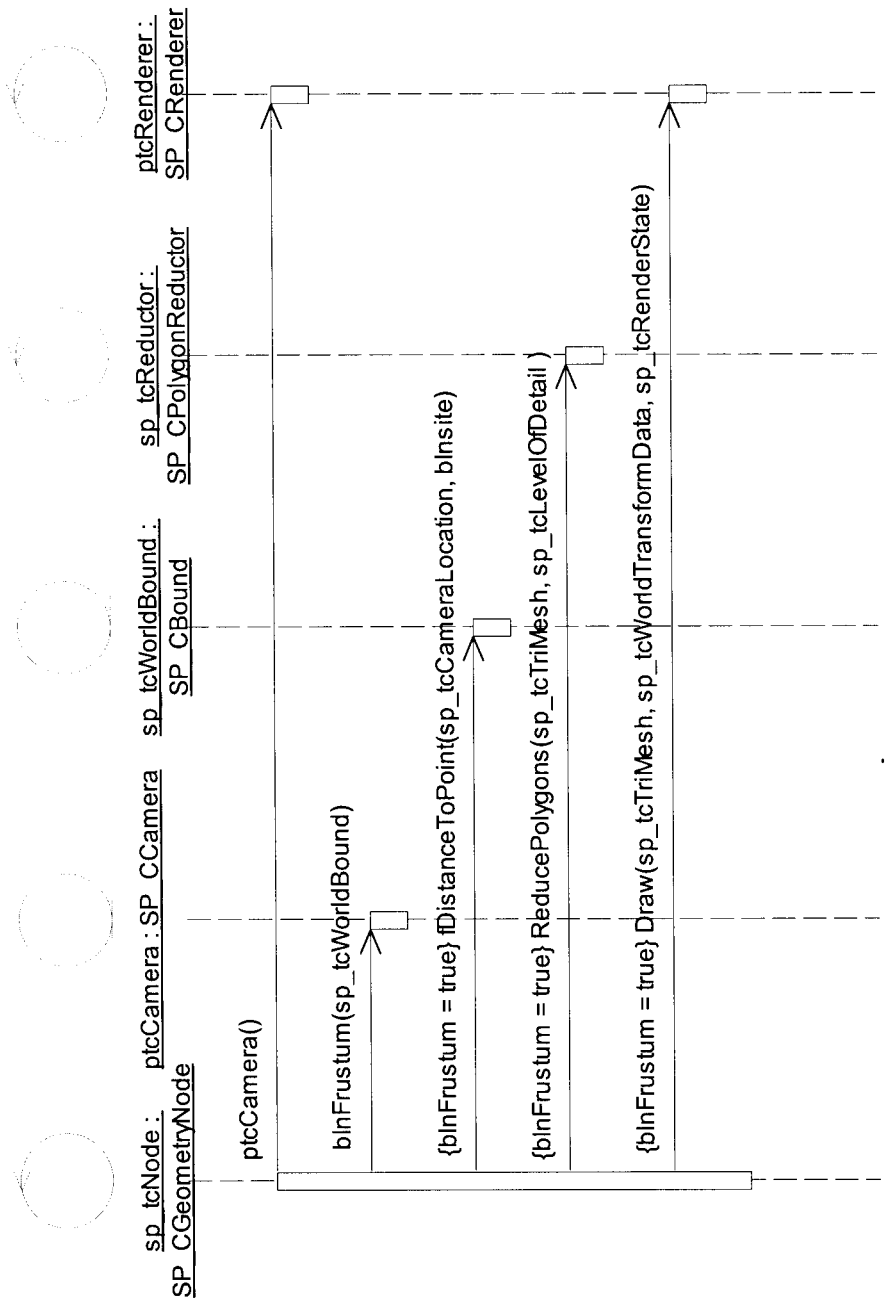

Fig. 64 Diagrama de Secuencia "Enviar a render (Extensión Geometría)".

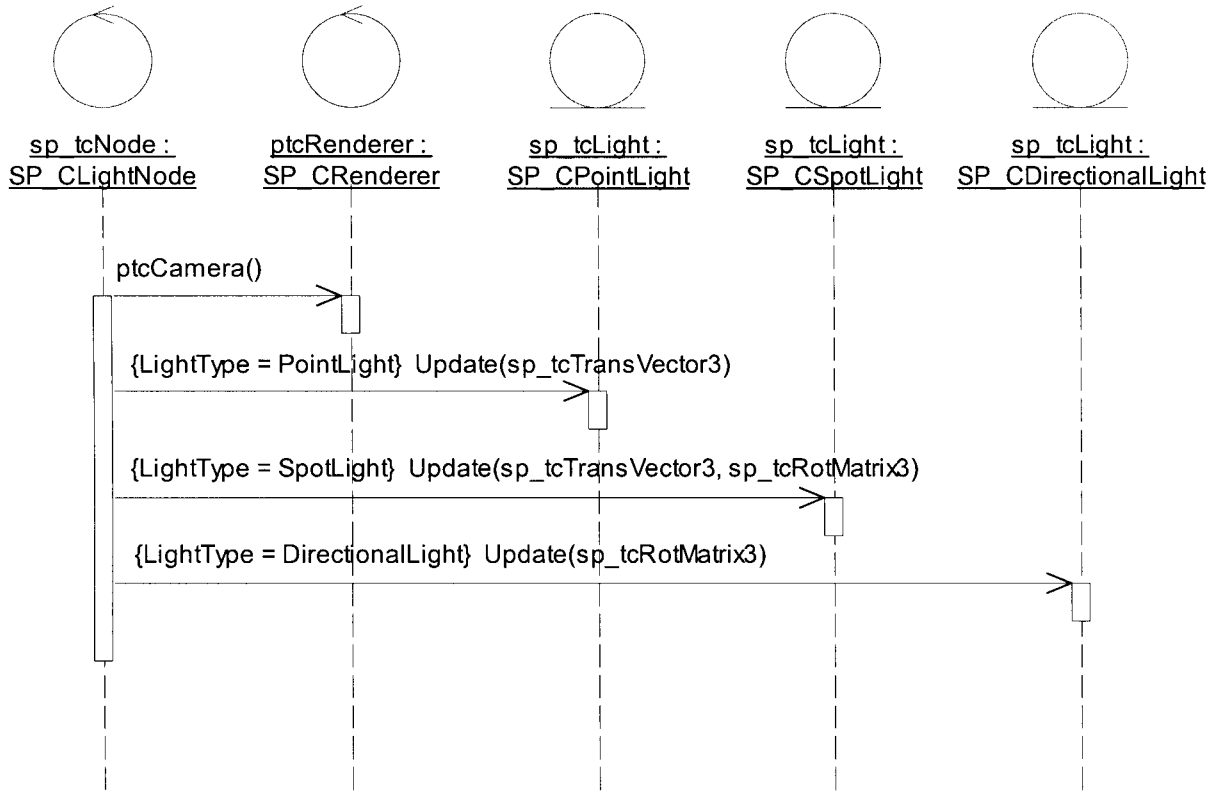

Fig. 65 Diagrama de Secuencia "Enviar a render (Extensión Luz)".

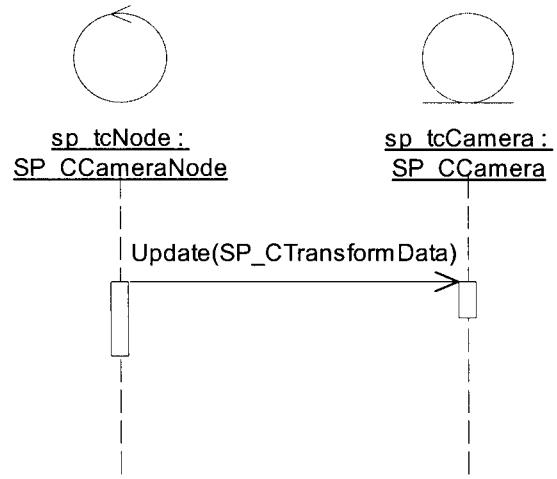

Fig. 66 Diagrama de Secuencia "Enviar a render (Extensión Cámara)".

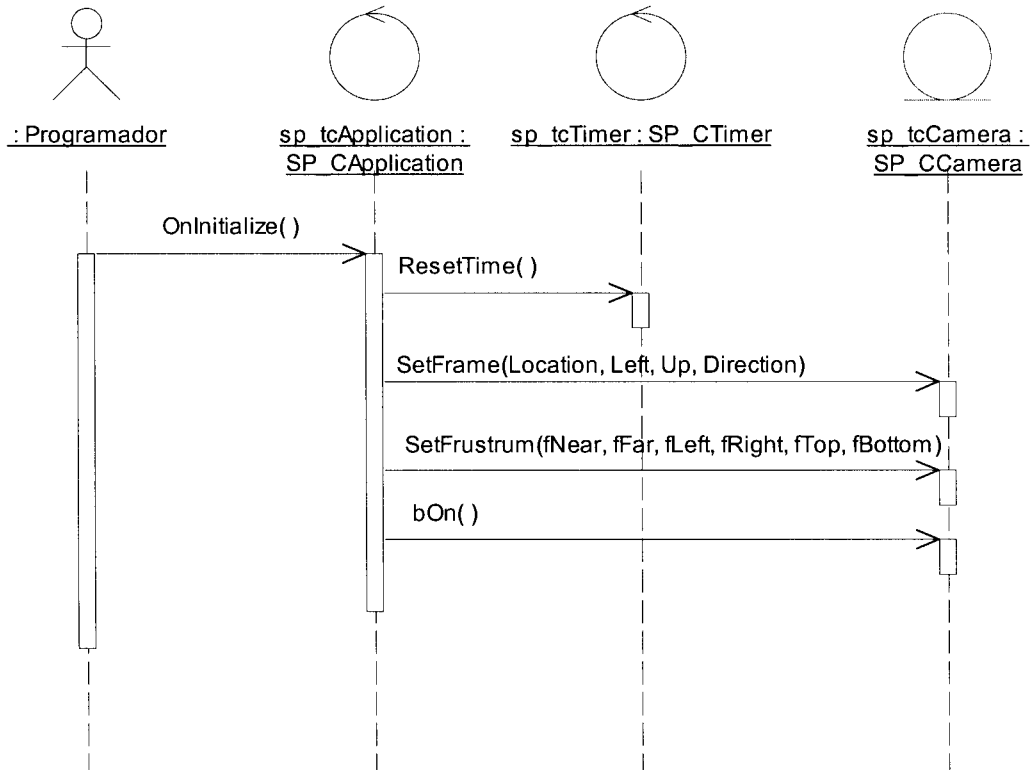

Fig. 67 Diagrama de Secuencia "Inicializar y actualizar escena (Sesión Inicializar)".

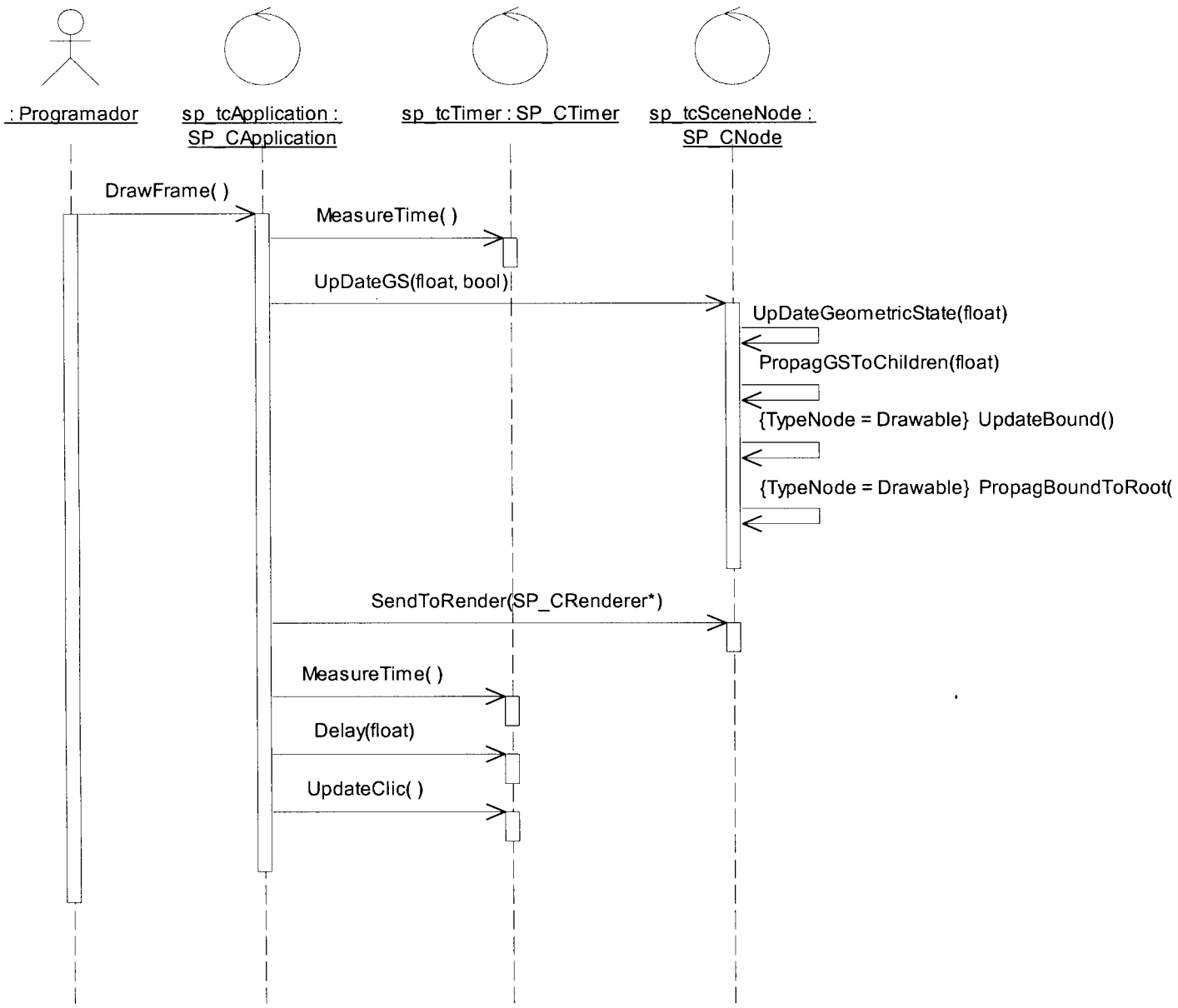

Fig. 68 Diagrama de Secuencia "Inicializar y actualizar escena (Sesión Actualizar)".

Conclusiones

Al concluir este capítulo se tiene concebido detalladamente el diseño completo del sistema y la secuenciación de pasos traducida a mensajes entre clases de los primeros casos de usos a desarrollar, con lo que se puede pasar a la etapa de implementación del proyecto.

5

Capítulo 5 Implementación del Sistema

Introducción

Esta etapa del proyecto constituye el paso del diseño de clases a la creación de componentes físicos, que se traducen en ficheros .cpp correspondiente a la implementación en C++. Además se elabora el diagrama de despliegue para el sistema.

5.1 Estándares de codificación

Se programará en inglés, debido que las palabras son simples, no se acentúan y es un idioma muy difundido en el mundo informático. Se respetarán los estándares de codificación para C++ (identado, uso de espacios y líneas en blanco, etc.).

Se nombrarán los ficheros .h y .cpp de la siguiente manera:
SPNameOfUnits.cpp

Se usará SP para identificar que pertenecen a la empresa (SIMPRO)

Los tipos se nombrarán siguiendo el siguiente patrón:
enum SP_EMyEnum {ME_VALUE, ME_OTHER_VALUE};
struct SP_SMyStruct {...}; /* variables miembros igual que en las Clases */
class SP_CClassName;

Las constantes se nombrarán con mayúsculas, utilizándose el "_" para separar las palabras: MY_CONST_ZERO = 0;

Las constantes de enumerados comenzarán con las iniciales del nombre del enumerado (ME_VALOR, en el ejemplo del enumerado).

Los nombres de las variables comenzarán con un identificado del tipo de dato al que correspondan, como se muestra a continuación. En el caso de que sean variables miembros de una clase, se le antepondrá el identificador "sp_" (en minúscula), si son globales se les antepondrá la letra "g", y en caso de ser argumentos de algún método, se les antepondrá el prefijo "arg_".

Declaración de variables:

/* tipos simples */
bool bVarName;

```
int iName;
unsigned int  uiName;
float fName;
char cName;
char* acName;       // arreglo
char* pcName;       // puntero
char** aacName;     // bidimensional
char** apcName;     // arreglo de punteros

bool sp_bMemberVarName; //variable miembro
char gcGlobalVarName;       //variable global, no se le antepone "sp_"

/* instancias de tipos creados, se usa "t" para diferenciarlos */
SP_EMyEnumerated teName;
SP_TMyStructure ttName;
SP_AMyArray taName;
SP_CClassName tcObjectName;
SP_CClassName* ptcName;    //se lee "puntero a tipo clase" o "puntero a objeto"
SP_CClassName* atcName;  //arreglo de objetos

SP_CClassName* sp_atcName;  // variable miembro de clase
```

En el caso de las funciones, se les antepondrá el identificador del tipo de dato de devolución, y en caso de no tenerlo (void), no se les antepondrá nada:

```
SP_CClassName (bool arg_bVarName, float & arg_fVarName);
// solamente los constructores y destructores comenzarán con "SP_"
// en el ejemplo anterior, véase las variables argumentos
~SP_CClassName;
bool bFunction1 (...);
int* piFunction2 (...);
```

```
SP_CClassName* ptcFunction3 (...);
Procedure4 (...);
```

Se usarán los "Gets" y "Sets" para acceder a los miembros de las clases, nombrándose como los demás métodos, pero con el nombre de la variable a la que se accede:

```
char sp_cMyVar;  //variable
char cMyVar();     //"get" (sin "sp_")
{
  return sp_cMyVar;
}
void MyVar(char* arg_cMyVar)  // "set"
{
 sp_cMyVar = arg_cMyVar;
}
```

5.2 Diagrama de despliegue

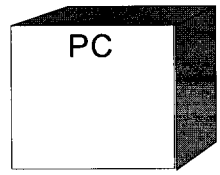

Fig. 69 Diagrama de Despliegue.

5.3 Diagramas de componentes

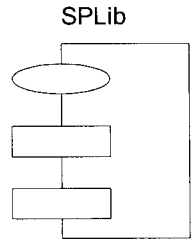

Fig. 70 Paquete de Componentes "SPLib".

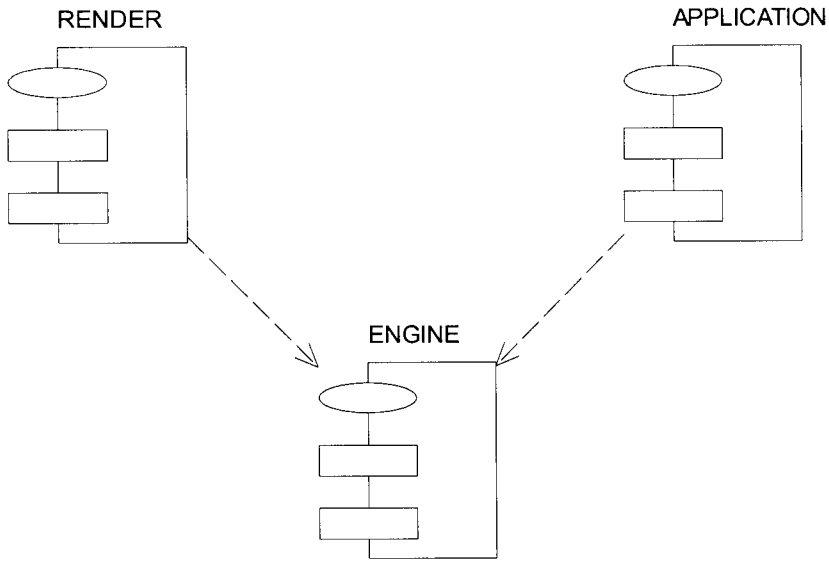

Fig. 71 SubPaquetes de Componentes "SPLib".

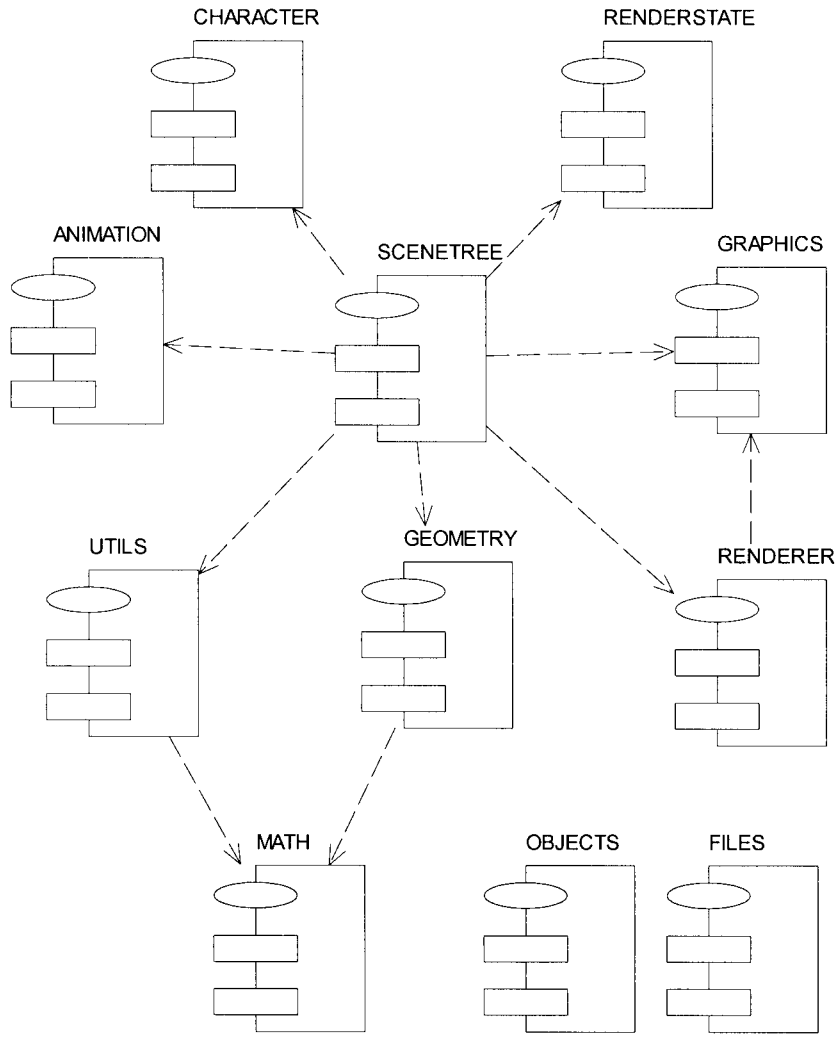

Fig. 72 SubPaquetes de Componentes "Engine".

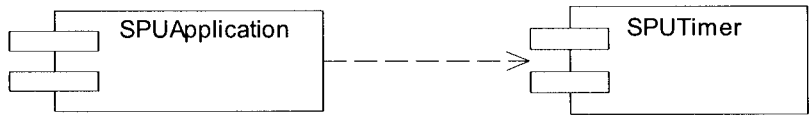

Fig. 73 Diagrama de Componentes "Application".

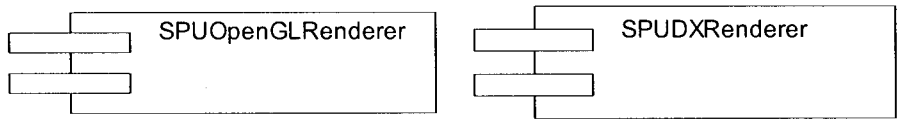

Fig. 74 Diagrama de Componentes "Render".

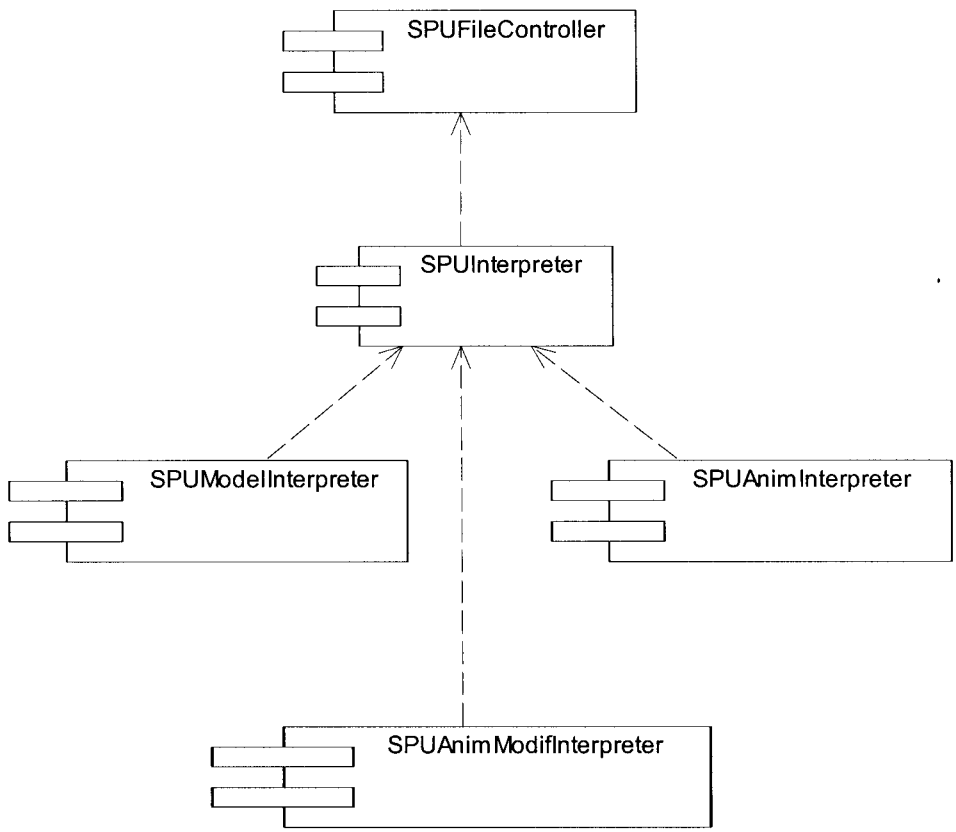

Fig. 75 Diagrama de Componentes "Files".

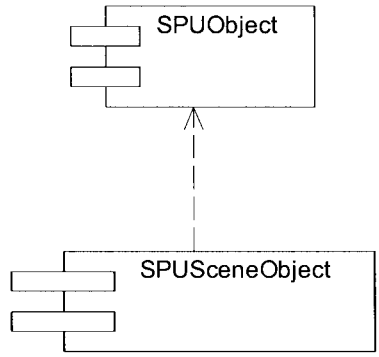

Fig. 76 Diagrama de Componentes "Objects".

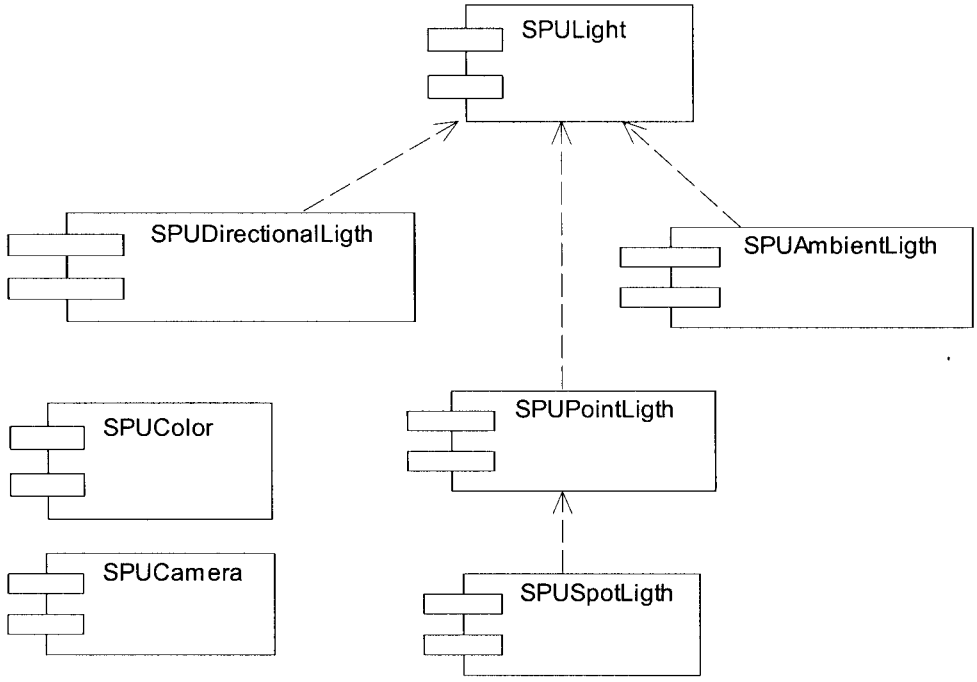

Fig. 77 Diagrama de Componentes "Graphics".

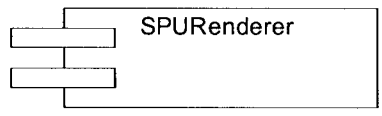

Fig. 78 Diagrama de Componentes "Renderer".

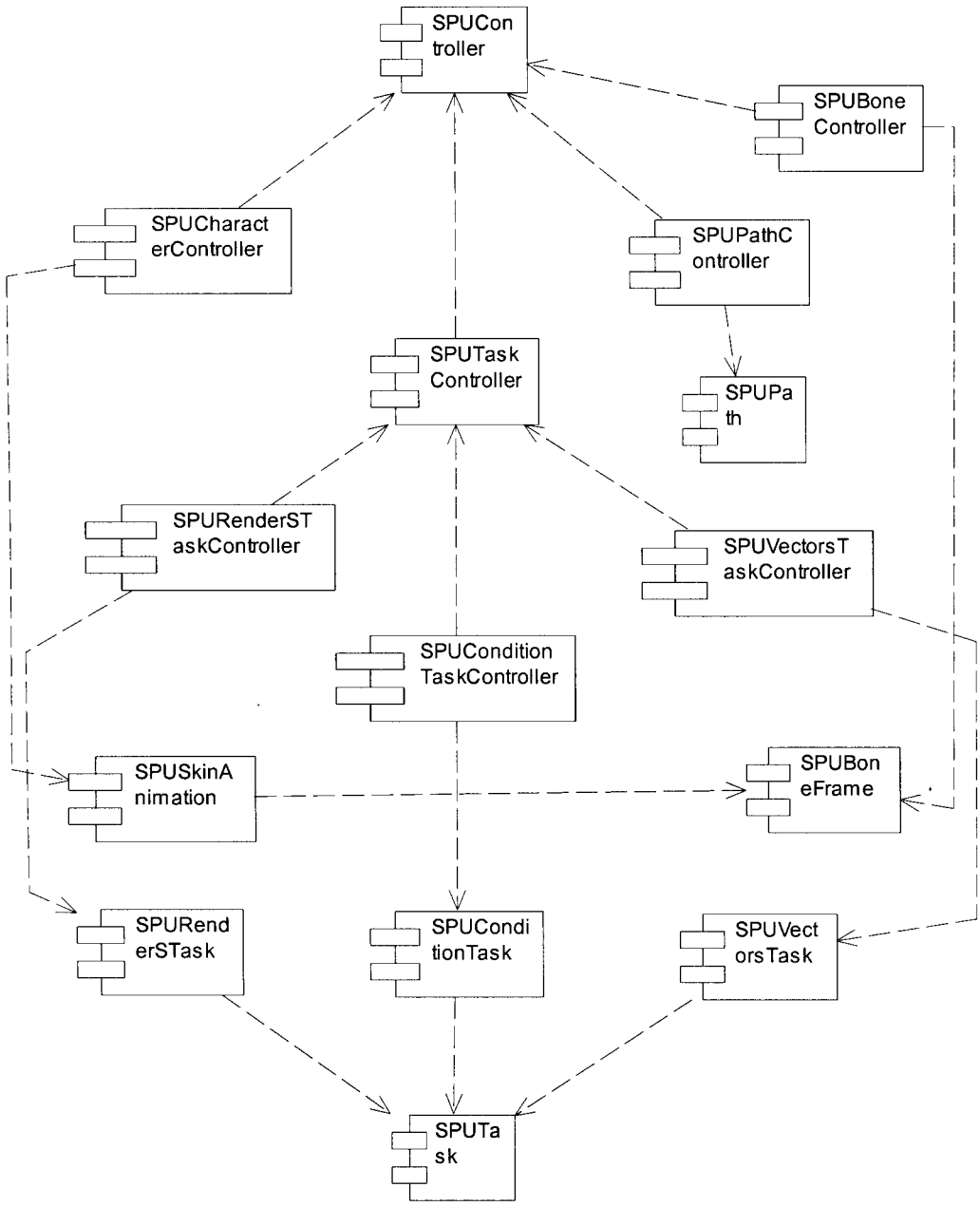

Fig. 79 Diagrama de Componentes "Animation".

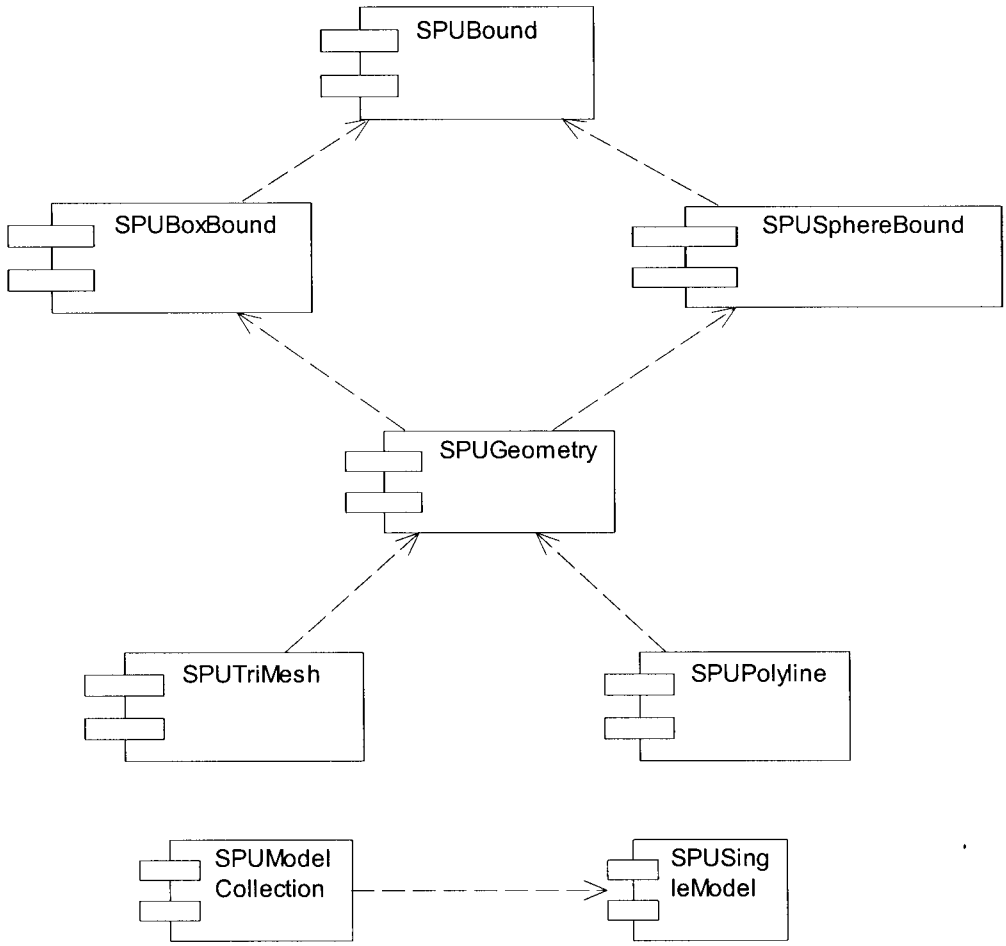

Fig. 80 Diagrama de Componentes "Geometry".

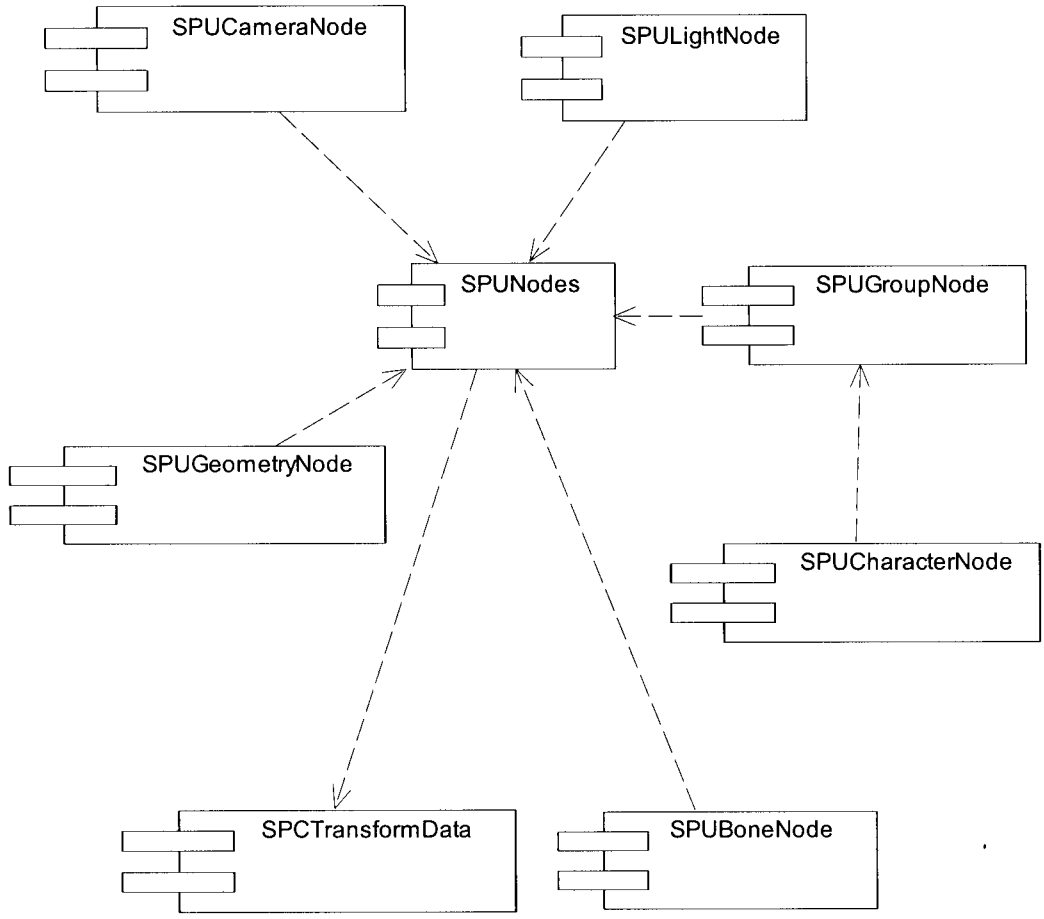

Fig. 81 Diagrama de Componentes "SceneTree".

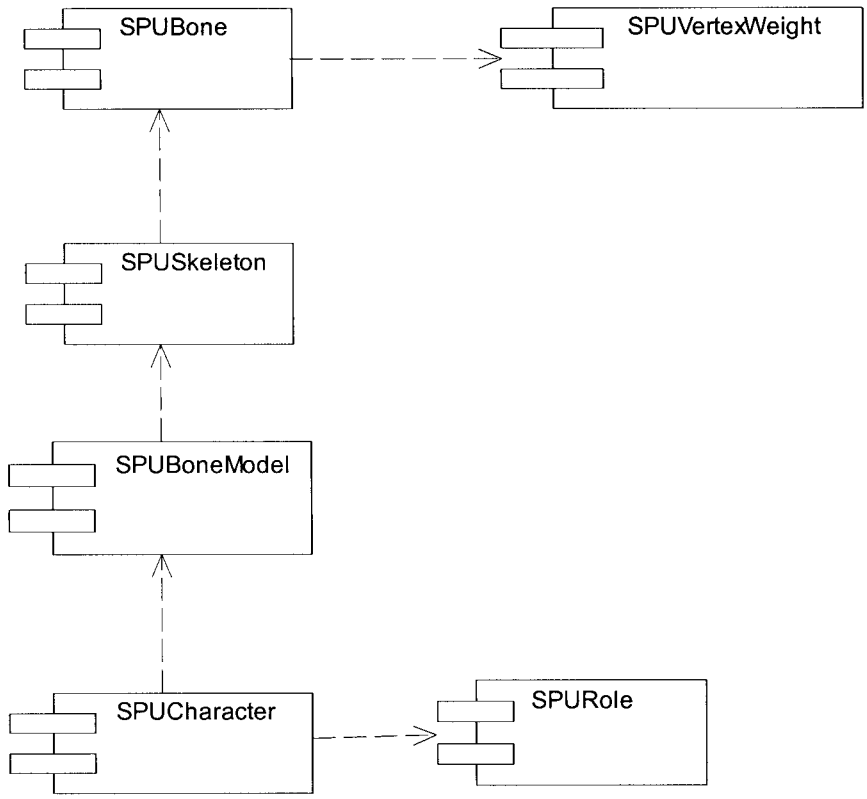

Fig. 82 Diagrama de Componentes "Character".

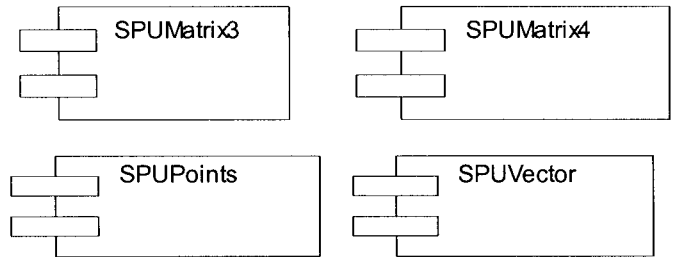

Fig. 83 Diagrama de Componentes "Math".

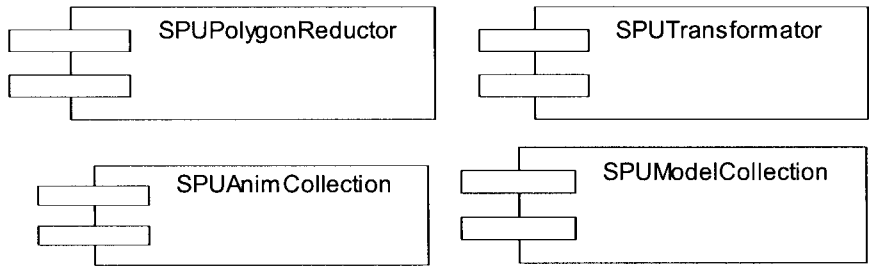

Fig. 84 Diagrama de Componentes "Utils".

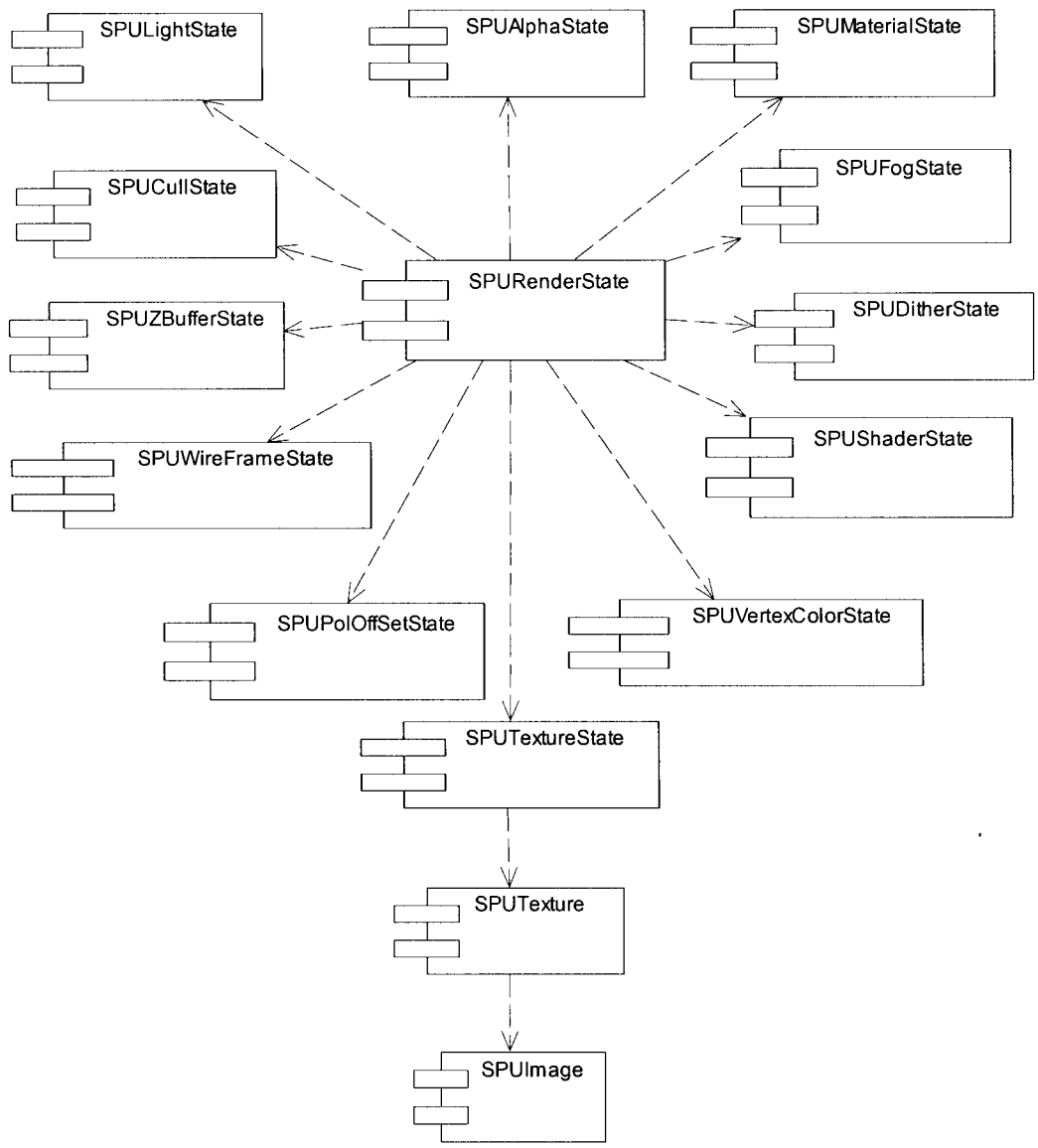

Fig. 85 Diagrama de Componentes "RenderState".

Conclusiones

En este momento se encuentra todo preparado para pasar a la etapa de programación de los casos casos de uso desarrollados en el primer ciclo. Como posibilidad adicional que brinda la herramienta Rational Rose, ya es posible generar el código fuente de los componentes relacionados con los casos de uso a desarrollar en el primer ciclo.

Conclusiones

Para el cumplimiento de los objetivos de este poyecto, en concordancia con las exigencias hechas por la entidad cliente, se requirió primeramente hacer un estudio de las técnicas, tecnologías y tendencias actuales en relación con el tema de los SRV en tiempo real. En el estudio se analizaron las características de éstos, y las deficiencias de algunas aplicaciones que los emplean y que se requiere sean erradicadas en este trabajo. Además, a partir de esa investigación se proponen las caracteristias técnicas de la solución.

Posteriormente se hizo la captura de los requisitos funcionales y no funcionales, y la agrupación de los primeros en los casos de uso del sistema. Se definió además una primera fase de desarrollo del proyecto, donde se realizarán solo una parte de los casos de uso para obtener la visualización de un modelo simple.

A continuación se transitó por las etapas de análisis, diseño e implementación utilizando los artefactos de RUP, donde surgieron y maduraron las clases, se realizaron los casos de uso a desarrollar en la primera fase y se creó el diagrama de componentes final que contendrá a las clases de la biblioteca.

Recomendaciones

Se recomiendan los siguientes aspectos al trabajo:

1- Desarrollar el módulo de animación facial de los personajes.
2- Desarrollar las técnicas de optimización en la visualización tanto para escenarios de interiores como de exteriores.
3- Incluir los algoritmos de detección de colisiones entre objetos.
4- Incluir algoritmos de seguimientos de terrenos.
5- Incluir la dinámica de los cuerpos.
6- Incorporar inteligencia artificial.
7- Implementar una herramienta visual que facilite el uso de las funcionalidades de la biblioteca gráfica.

Referencias bibliográficas

Libros

[1]. ASTLE, Dave y HAWKING, Kevin.
OpenGL Game Programming.
Prima Tech Publishing. USA. 2001.
[1.1]. Capítulo 1: *The exploration begins: OpenGL and DirectX.*
[1.2]. Capítulo 2: *Using Windows with OpenGL.*
[1.3]. Capítulo 3: *An Overview of 3D Grafics Theory.*
[1.4]. Capítulo 15: *Special Effects.*
[1.5]. Capítulo 19: *Physics Modeling with OpenGL. Modeling the Real World.*
[1.6]. Capítulo 20: *Building a Game Engine.*

[2]. BIRN, Jeremy.
Digital lightning and rendering.
New Riders Publishing. USA. 2000.
[2.1]. Capítulo 6: *Colors.*

[3]. LUNA, Frank D
Introduction to 3D Game Programming with DirectX.
WordWare Publishing, Inc. USA. 2003.
[3.1]. Capítulo 6: *Texturing.*
[3.2]. Capítulo 13: *Basic Terrain Rendering.*

[4] MAESTRI, George.
Character animation 2.
Volume 1: *Essential techniques.* New Riders Publishing, USA. 1999
[4.1]. Capítulo 6. *Skeletons and mesh deformation.*

[5]. Sense8 CORPORATION TEAM

WorldToolkit Release 7, Reference Manual.

Sense8 Corporation (www.sense8.com). USA. 1997.

[5.1]. Capítulo 3. *Scene Graphs.*

[6]. Sense8 CORPORATION TEAM

WorldToolkit Release 8, Technical Overview.

Sense8 Corporation (www.sense8.com). USA . 1998.

[6.1]. Capítulo 3. *Scene Graphs.*

[7]. Sense8 CORPORATION TEAM

WorldToolkit Release 9, Reference Manual.

Sense8 Corporation (www.sense8.com). USA. 1999.

[7.1]. Capítulo 4. *Scene Graphs.*

[7.2]. Capítulo 5. *Geometries.*

[7.3]. Capítulo 7. *Materials.*

[7.4]. Capítulo 11. *Lights.*

Libros digitales

[8]. CHOVER, Miguel.

Informática Gráfica.

http://www3.uji.es/~jromero/grafica (2004).

[8.1]. Capítulo 2. *Modelado geométrico I.*

http://www3.uji.es/~jromero/grafica/Documentos/2-Modelado1IG35.pdf

[8.2]. Capítulo 3. *Transformaciones.*

http://www3.uji.es/~jromero/grafica/Documentos/3-TransformacionesIG35.pdf

[8.3]. Capítulo 5. *Visibilidad.*

http://www3.uji.es/~jromero/grafica/Documentos/5-VisibilidadIG35.pdf

[8.4]. Capítulo 8. *Modelado geométrico II.*

http://www3.uji.es/~jromero/grafica/Documentos/8-Modelado2IG35.pdf

[8.5]. Capítulo 9. *Aplicaciones de la Informática Gráfica.*
http://www3.uji.es/~jromero/grafica/Documentos/9-AplicacionesIG35.pdf

[9]. LANDER, Jeff.
Skin Them Bones: Game Programming for the Web Generation.
Game Developer Magazine. USA. May, 1998.
http://www.darwin3d.com/gamedev/articles/col0598.pdf (2004)

[10]. RUIZ, David y otros.
Animación en "Art of Illusion".
http://www.dccia.ua.es/dccia/inf/asignaturas/RG/trabajos/trabajo-david-ruiz.pdf
(2004)

[11]. SUÁREZ, Pamela.
Animación y Visualización de Fenómenos Naturales.
http://mail.udlap.mx/~tesis/lis/suarez_r_pk/ (2004).
[11.1]. Capítulo III. *Animación por computadoras.*
http://www.pue.udlap.mx/~tesis/lis/suarez_r_pk/capitulo3.pdf

[12]. UNIVERSITAT JAUME I
Curso de Multimedia.
http://www4.uji.es/~belfern/IS34/ (2004).
[12.1]. Capítulo 9: *Animación.*
http://www4.uji.es/~belfern/IS34/Documentos/Teoria/Capitulo9.pdf

[13]. UNIVERSITAT JAUME I
Motores de juegos
http://ima.udg.es/iiia/GGG/TIC2001-2416-C03-01/docs/EnginesUJI.pdf
(2004).

[14]. UNIVERSIDAD DE VALENCIA.
http://informatica.uv.es (2004)
[14.1]. *Tema 3: Sistemas de Visualización en Tiempo Real. Visualización por Hardware*
http://informatica.uv.es/iiguia/2000/AIG/web_teoria/tema3.pdf
[14.2]. *Tema 6: Técnicas para simulación en tiempo real.*
http://informatica.uv.es/iiguia/2000/AIG/web_teoria/tema6.pdf

Sitios web

[15]. 3D-GameStudio
http://www.conitec.net/a4info.htm (2004)

[16]. BioVision Incorporated
http://www.biovision.com (2004)

[17]. CONTACTO Magazine
Realidad Virtual, de los Programas Militares a Videojuegos e Internet.
http://www.contactomagazine.com/realidadvirtual0417.htm (2004)

[18]. KAUFMANN , Morgan.
Computer animation: algorithms and techniques.
http://www.cis.ohio-state.edu/~parent/book/outline.html (2004)
[18.1] Capítulo 2. *Recording Techniques and Animation Hardware.*
http://www.cis.ohio-state.edu/~parent/book/Rcrd.html#RealTime

[19]. LASSETER, John.

Tricks to Animating Characters with a Computer

http://www.siggraph.org/education/materials/HyperGraph/animation/character_animation/principles/lasseter_s94.htm, (2004)

[20]. PUTZ, Michael y HUFNAGL Klaus.

Character Animation for Real-time Applications

http://www.cg.tuwien.ac.at/studentwork/CESCG/CESCG-2002/ (2004)

[21]. RTX 3ds max exporter and viewer

http://rtx.cluetec.com/ (2004)

[22]. VJuegos: Comunidad Iberoamericana de Desarrolladores de juegos.

http://www.vjuegos.org (2004)

[22.1]. *Análisis de Lenguajes de Programación*

http://www.vjuegos.org/modules.php?name=Content&pa=showpage&pid=4

[22.2]. *Sincronización de Juegos*

http://www.vjuegos.org/modules.php?name=Content&pa=list_pages_categories&cid=6

Bibliografía consultada

- ABRASH, Michael. *Graphics Programming Black Book.* Dr. Dobb's Journal, 2001.
- ASTLE, Dave & HAWKINS, Kevin. *OpenGL Game Programming.* Premier Press, 2002.
- BETHKE, Erik. *Game Development and Production.* Wordware Publishing, Inc., 2003.
- BUSS, Samuel R. *3D Computer Graphics: A Mathematical Introduction With Opengl.* Cambridge University Press, 2003.
- DEMPSKI, Kelly. *Real-Time Rendering Tricks and Techniques in DirectX.* Premier Press, 2002.
- EBERLY, David H. *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics.* Academic Press, 2001.
- LEVER, Nik. *Real-time 3D Character Animation with Visual C++.* Planta Tree, 2002.
- MAESTRI, George. *Character Animation 2, volumen 1 – essential techniques.* New Riders, 1999.
- MOLLER, Tomas & HAINES, Eric. *Real-Time Rendering.* AK Peters, Ltd., Primera Edición, 1999.
- PARENT, Rick. *Computer Animation: Algorithms and Techniques.* Morgan Kaufmann Publishers, Primera Edición, 2001.
- Recopilación de autores. *Game Programming Gems.* Charles River Media, 2000.
- SHIRLEY, P. & SHIRLEY, Peter. *Fundamentals of Computer Graphics.* AK Peters Ltd, Primera Edición, 2002.
- WATT, Alan & POLICARPO, Fabio. *3D Games: Real-Time Rendering and Software Technology, Volume 1.* Addison Wesley Publishing Company, 2000.

Apéndices

Organigrama de SIMPRO

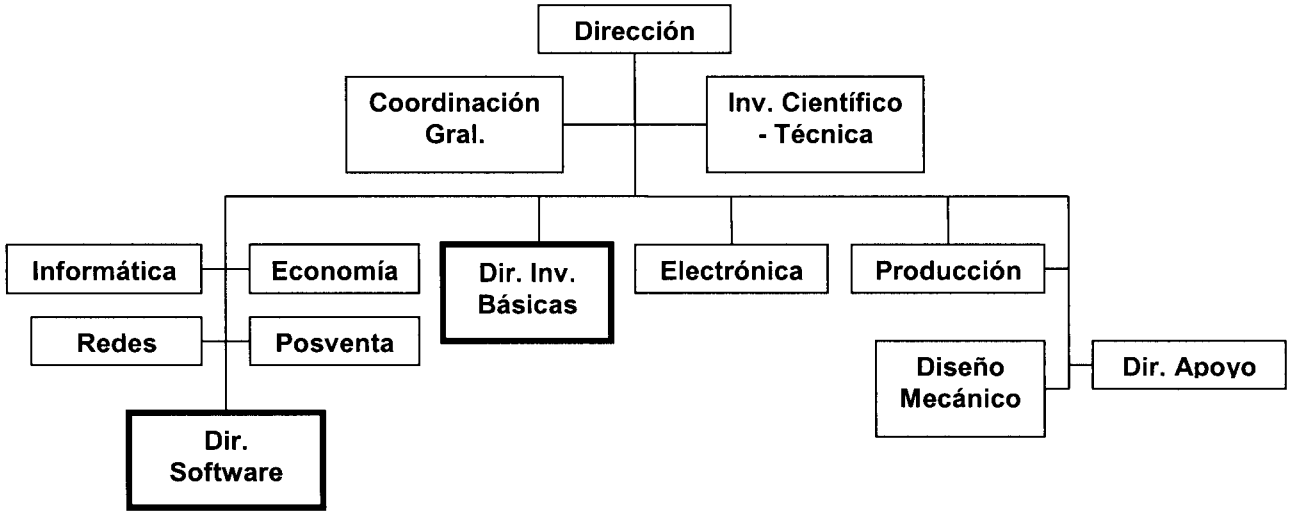

Fig. 86 Organigrama de SIMPRO

En negritas el campo de acción que abarco el proyecto.

Glosario de abreviaturas

API: Application Programmer's Interface, (interfaces para programadores de aplicaciones).

BSP: Binary Space Partition (Partición binaria del espacio).

CAD: Computer Aided Design (Diseño Asistido por Computadoras): herramientas gráficas que permiten diseñar prototipos y evaluarlos antes de construirlos.

COM: Component Object Model.

EG: Estado Geométrico.

ER: Estados de *Render*.

GDI: Grafics Device Interface (Interfaces de Dispositivos Gráficos).

GLUT (GLU): OpenGL Utility ToolKit (Herramientas y Utilidades de OpenGL).

HAL: Hardware abstraction layer (capa de abstracción de Hardware).

HEL: Hardware emulation layer (capa de simulador de Hardware).

HW: Hardware.

LOD: Level-of-detail (nivel de detalle).

PVS: Potentially visible sets (espacios cerrados potencialmente visibles): Se basa en la evaluación de una matriz de conectividad para cada hoja del BSP desde puntos aleatorios en la escena. Cuando se llega a una hoja esta sólo se visualiza si está en la matriz de conectividad del nodo donde está el punto de vista.

RGB: Red Green Blue. Los colores RGB consisten en tres números, que representan los niveles de rojo, verde y azul, respectivamente, conocidos como colores aditivos primarios.

SRV: Sistemas de Realidad Virtual.

SW: Software.

VF: Volumen Frontera.

WGL: Wiggle.

WTK: WorldToolKit. Entorno para el desarrollo de aplicaciones de Realidad Virtual.

Glosario de términos

A:

Aliasing: las líneas, especialmente las que están casi horizontales o verticales, aparecen dentadas o irregulares debido a su representación por *pixels*. Este escalonamiento es llamado *aliasing*.

Animación: simulación de un movimiento creada por la muestra de una serie de imágenes o fotogramas.

Antialiasing: técnicas que se utilizan para reducir el efecto de *aliasing*.

*****

C:

Comportamiento (behavior): cambio de atributos o características de los objetos visibles y no visibles del mundo virtual y que cambian el aspecto visual de la escena logrando una animación.

Cono de visión de la cámara: (ver volumen de visión de la cámara).

Coplanar: grupo de puntos o líneas que están en un plano común.

Cualidades físicas y emocionales: cualidades de los personajes del mundo real que se simularán con los personajes virtuales a través de la manera de ejecutar sus acciones. Son estas: tipo de personaje, temperamento, rango de velocidades de acciones, estado de acción y estado de actividad (todas definidas en este glosario.

*****

E:

Emoción: característica de personajes del mundo real que se simula con los personajes del mundo virtual, y que permite añadirle variaciones a las acciones de los personajes teniendo en cuenta, por ejemplo, su estado anímico.

Engine: programa orientado a la creación de otros softwares.

Entorno sintético: mundo virtual.

Estado de actividad: estado del personaje que simula el desgaste físico de personajes reales a través de la velocidad de la animación.

Estado de espera del personaje: estado en que se encuentran los personajes que deberían estar "inactivos", y que busca darle realismo al simulador evitando que parezcan objetos inanimados.

Estado geométrico: información posicional (posición, orientación y escala) de los objetos de la escena.

Estado de render: información de estados de dibujo utilizada por el *renderer* para representar las mallas de la escena.

Estereoscopía: apreciación de las diferentes distancias y volúmenes en el entorno dando sensación de profundidad, lejanía o cercanía de los objetos.

*****

F:

Fotograma: (ver *frame*)

Frame: cada uno de las imágenes que componen una animación.

Framerate: número de *frames* por segundo.

Frustum de cámara: conjunto de planos que definen el volumen de visión de la cámara.

*****

G:

Gouraud shading: técnica usada para dar efectos de sombra a objetos 3D compuestos de polígonos, consistente en interpolar intensidades de luces en los vértices de cada cara de polígono, mostrando una superficie lisa.

Grafo de escena: estructura jerárquica donde se almacenan y organizan los objetos de la escena para el control de su información y determinación de la visibilidad, donde las hojas contienen los objetos dibujables de la escena.

*****

I:

Interpolación: algoritmo matemático que a partir de varios puntos en el espacio, describe una función que contiene a los puntos intermedios.

*****

M:

Malla: forma de representar un modelo a partir de polígonos. Colección de vértices, aristas y polígonos conectados de forma que cada arista es compartida como máximo por dos polígonos.

Mapa de texturas: correspondencia entre vértices de una textura y los vértices del modelo.

Material: combinación de luces y colores usados para definir una apariencia.

Matrices de transformación: matrices definidas para calcular nuevas coordenadas a partir de coordenadas existentes según una determinada transformación gráfica (rotación, trasladación, escalado y reflexión).

Modelo: prototipo para la animación.

Motor de simulador: armazón orientada a objeto con las funcionalidades básicas para construir simuladores.

*****

N:

Normal: (ver vector normal).

*****

P:

Path: lista de vectores que definen una trayectoria a través de la escena.

Paso de interpolación: distancia entre los puntos a obtener en una interpolación.

Personaje: actor de la escena de un mundo de realidad virtual, que soporta acciones (como un tipo de comportamiento), y que tienen entre sus atributos, cualidades físicas y emocionales que serán usadas a la hora de ejecutar las acciones, así como determinados roles (ver rol).

Pivote de rotación: eje respecto al cual se le aplica la rotación a un cuerpo.

Píxel: abreviatura de "picture element". Es la menor unidad de información de una imagen digital.

Propiedades físicas: propiedades de los objetos del mundo real que se simularán con los objetos virtuales a través de la manera de ejecutar sus tareas, por ejemplo, la masa.

Proyección: conversión de coordenadas 3D del mundo a las coordenadas 2D de un plano.

Proyección paralela (u ortográfica): es aquella en que todos los objetos mantienen sus dimensiones en la proyección plana sin importar cuan lejos estén en el mundo.

Proyección perspectiva: determina los tamaños de los objetos basándose en la distancia de los objetos al plano de proyección.

*****

Q:

Quad: *quadrangle*, abreviatura de cuadrángulo.

*****

R:

Rango de velocidades de acciones: intervalo de velocidades que describen las posibilidades físicas de un tipo de personaje.

Renderer: (como se usa en este documento): componente del sistema gráfico de la computadora, responsable de dibujar los triángulos de un modelo utilizando la información de estados de dibujo, llamada estados de *render*.

Rendering: crear en forma automática una imagen de acuerdo al modelo tridimensional que existe en el ordenador.

Rendering pipeline: serie de pasos que se siguen para hacer el *rendering*.

Renderizar: (ver *rendering*)

Rol: papel que trae consigo un grupo de acciones y que formará parte de los atributos de un personaje animado.

Rotoscopía: técnica de animación que consiste en capturar un movimiento real, y utilizar esa información para mover un diseño generado por ordenador.

*****

S:

Sistema de Realidad virtual: sistema informático interactivo que ofrece una percepción sensorial al usuario de un mundo tridimensional sintético que suplanta al real.

Skinning: técnica de deformación de malla por huesos con múltiples pesos.

Stitching: técnica de deformación de malla por huesos.

*****

T:

Tarea: se usan para asignar comportamientos a los objetos individualmente.

Temperamento: rasgo de la personalidad que se simula a través de la velocidad de las animaciones.

Texel: par (u,v) que identifica un elemento de la textura.

Textura: imagen que sirve de "piel" a los modelos en un mundo virtual.

Texturizar: aplicar de texturas.

Tubería de rendering: (ver *Rendering pipeline*)

V:

Vector: cantidad que expresa magnitud y dirección.

Vector de traslación: vector que indica el sentido y dirección en que se moverá un objeto.

Vector normal: vector cuyos puntos están en dirección perpendicular a una superficie.

Vector unidad: vector de longitud 1.

Volumen frontera: figura geométrica que delimita el espacio ocupado por un cuerpo.

Volumen de visión de la cámara: sección de espacio 3D visible por la cámara, definido por seis planos: cercano, lejano, izquierdo, derecho, arriba y abajo.

Diagramas de clases de análisis

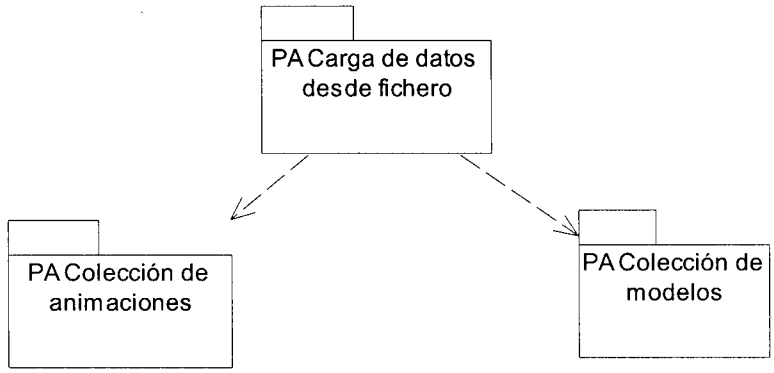

Fig. 87 Diagrama de Paquetes. Paquete "PA Almacenamiento de datos".

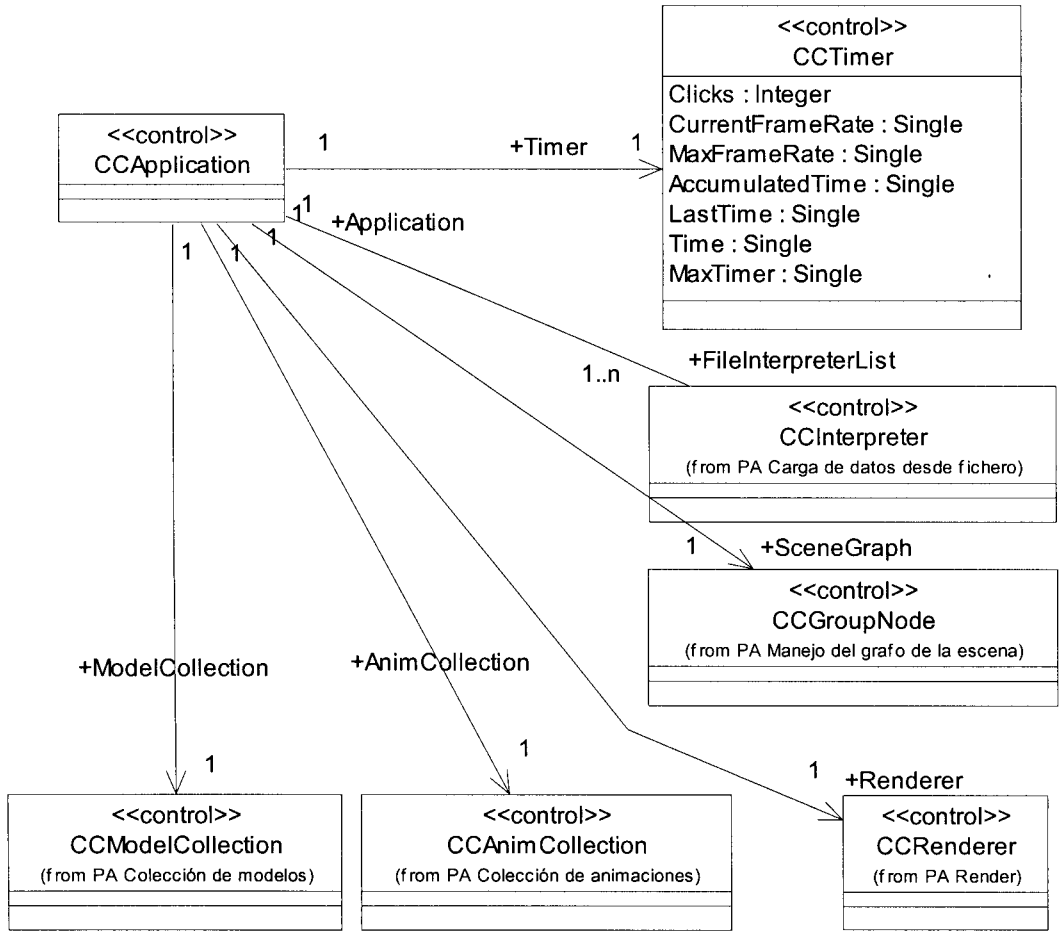

Fig. 88 Diagrama de Clases de Análisis "Aplicación".

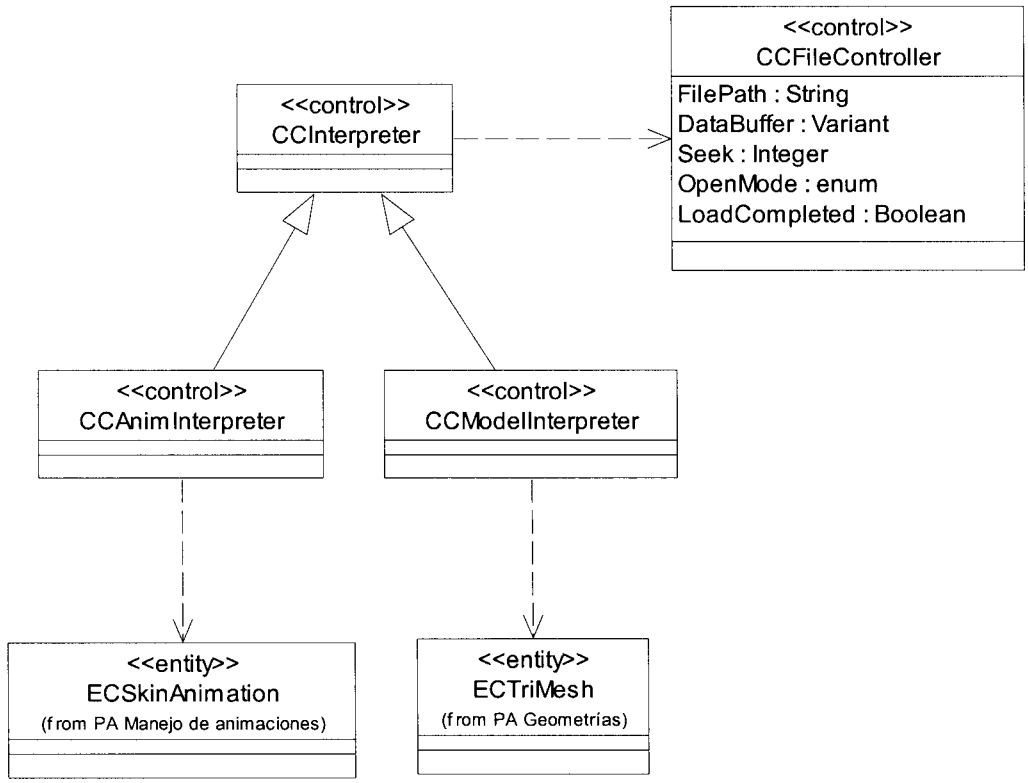

Fig. 89 Diagrama de Clase de Análisis "Carga de datos desde ficheros".

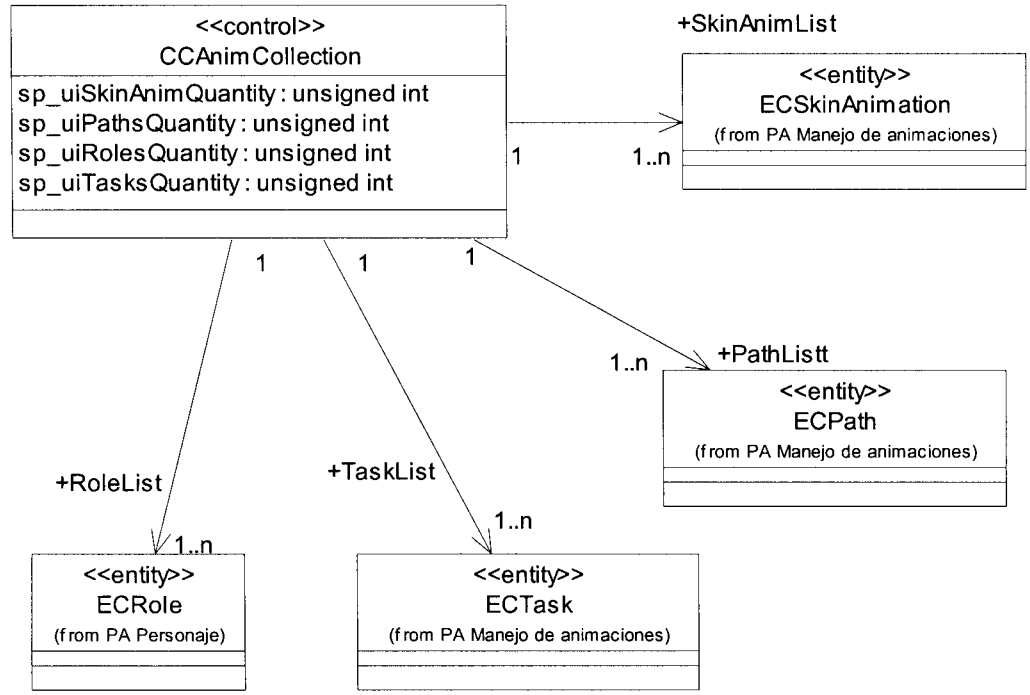

Fig. 90 Diagrama de clase de Análisis "Colección de animaciones".

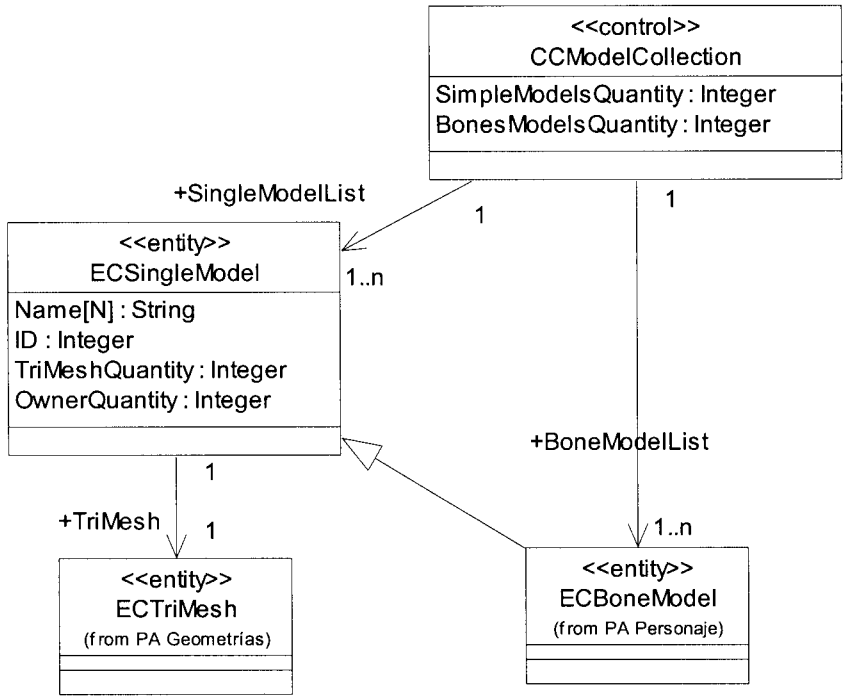

Fig. 91 Diagrama de Clases de Análisis "Colección de modelos".

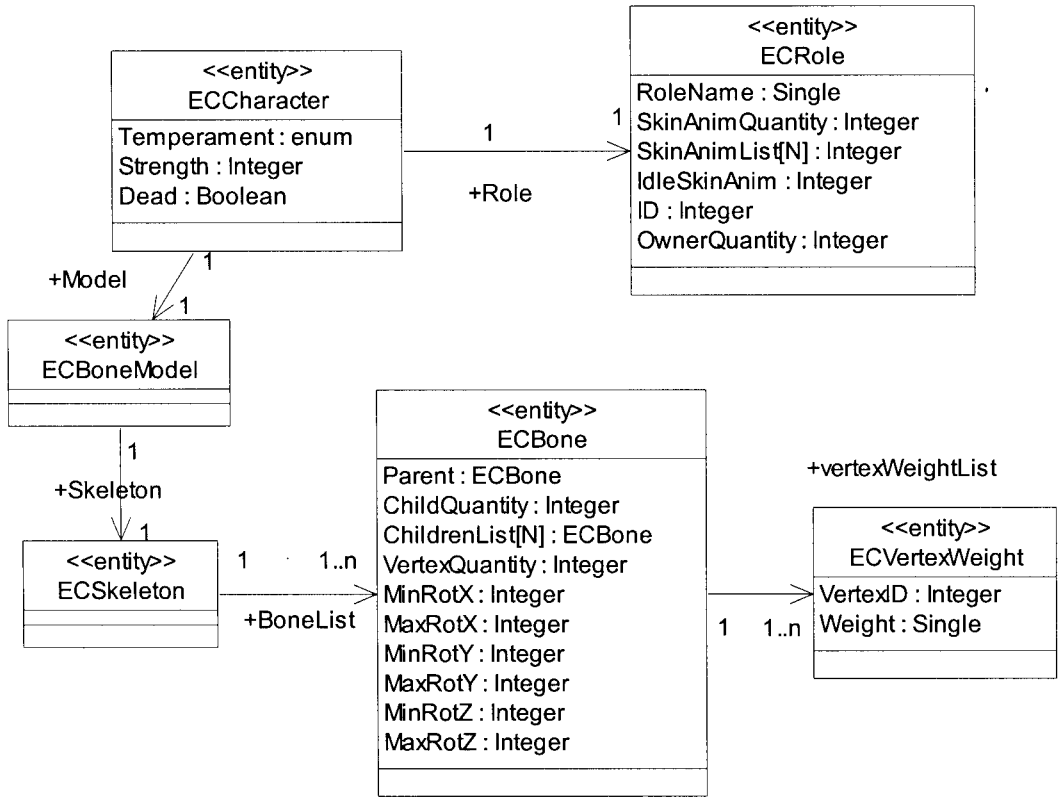

Fig. 92 Diagrama de Clases de Análisis "Personaje".

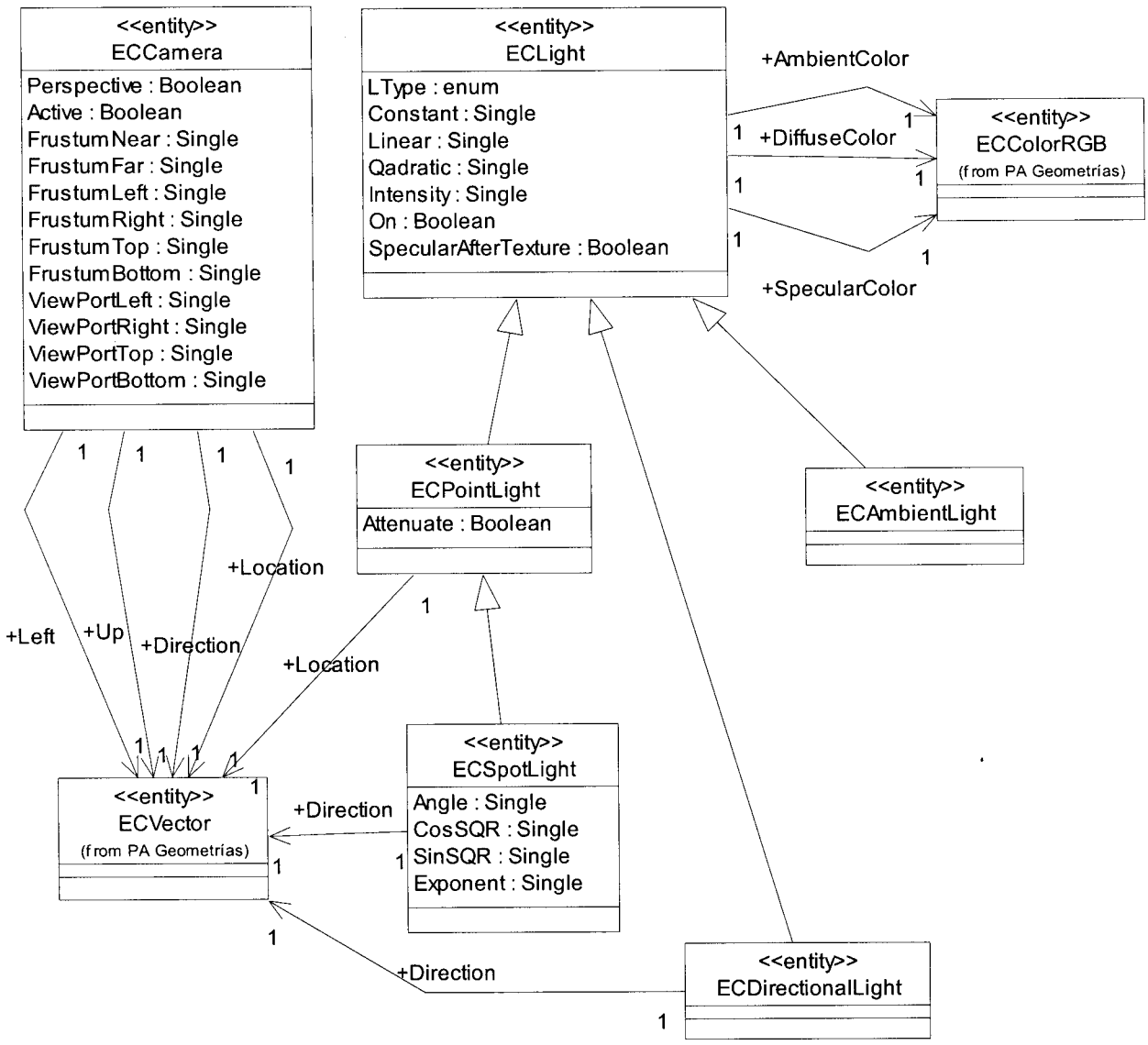

Fig. 93 Diagrama de Clases de Análisis "Luces y cámaras".

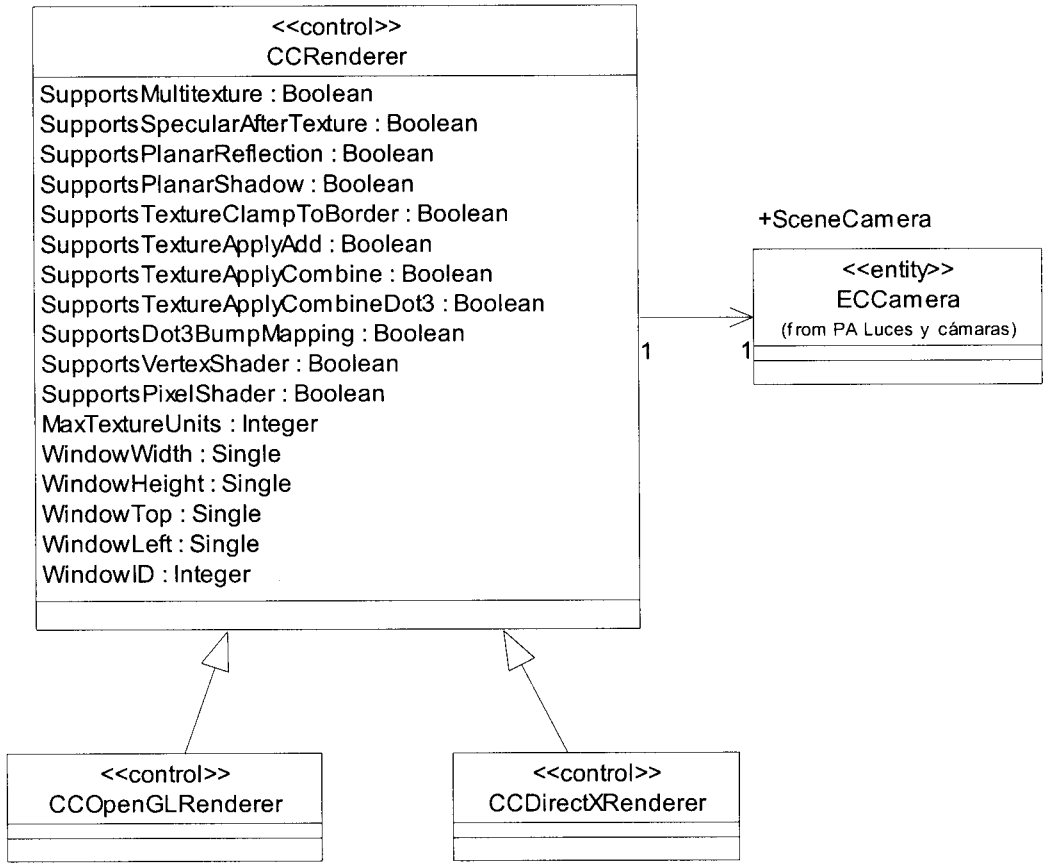

Fig. 94 Diagrama de Clases de Análisis "Render".

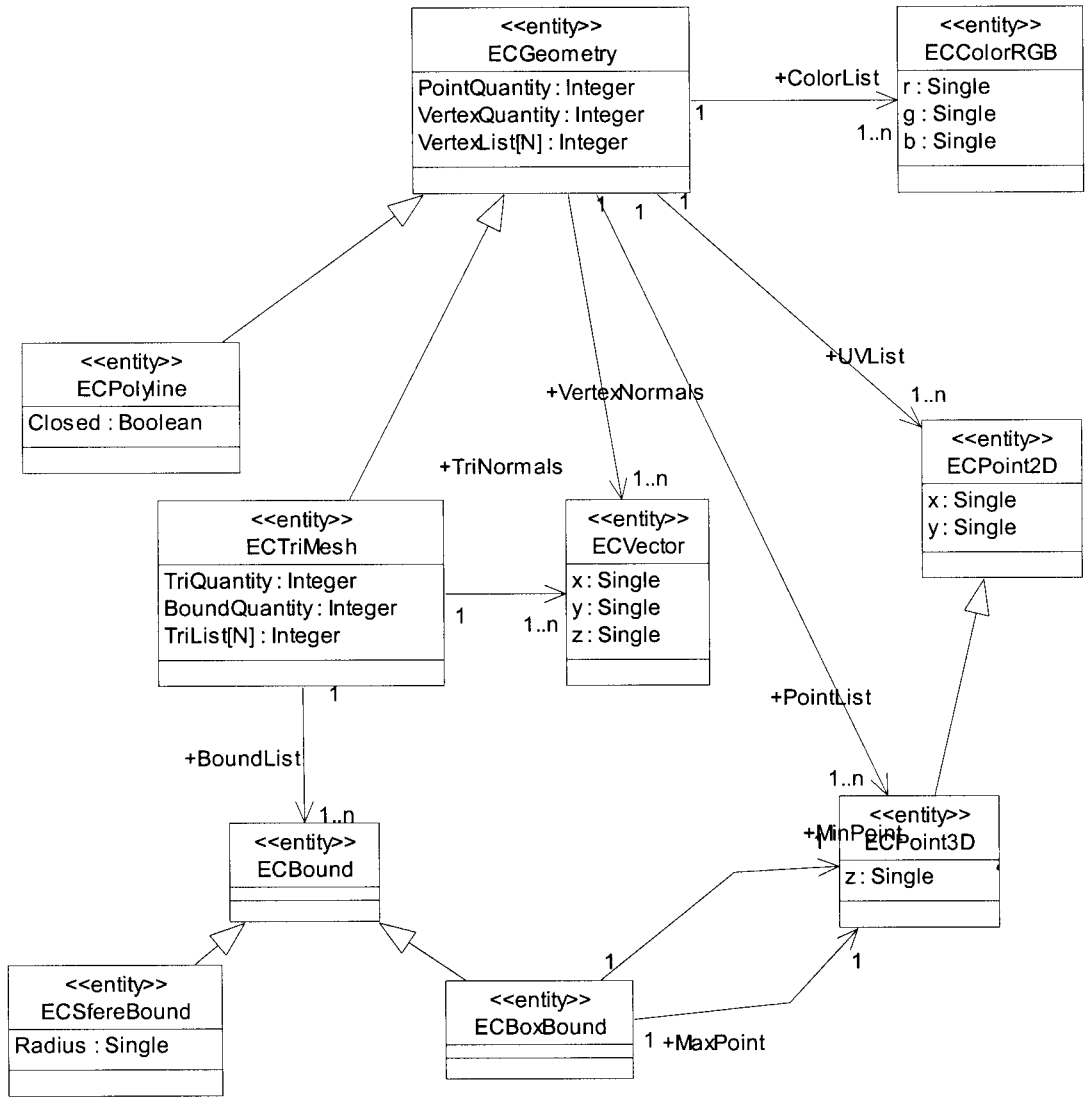

Fig. 95 Diagrama de Clases de Análisis "Geometrías".

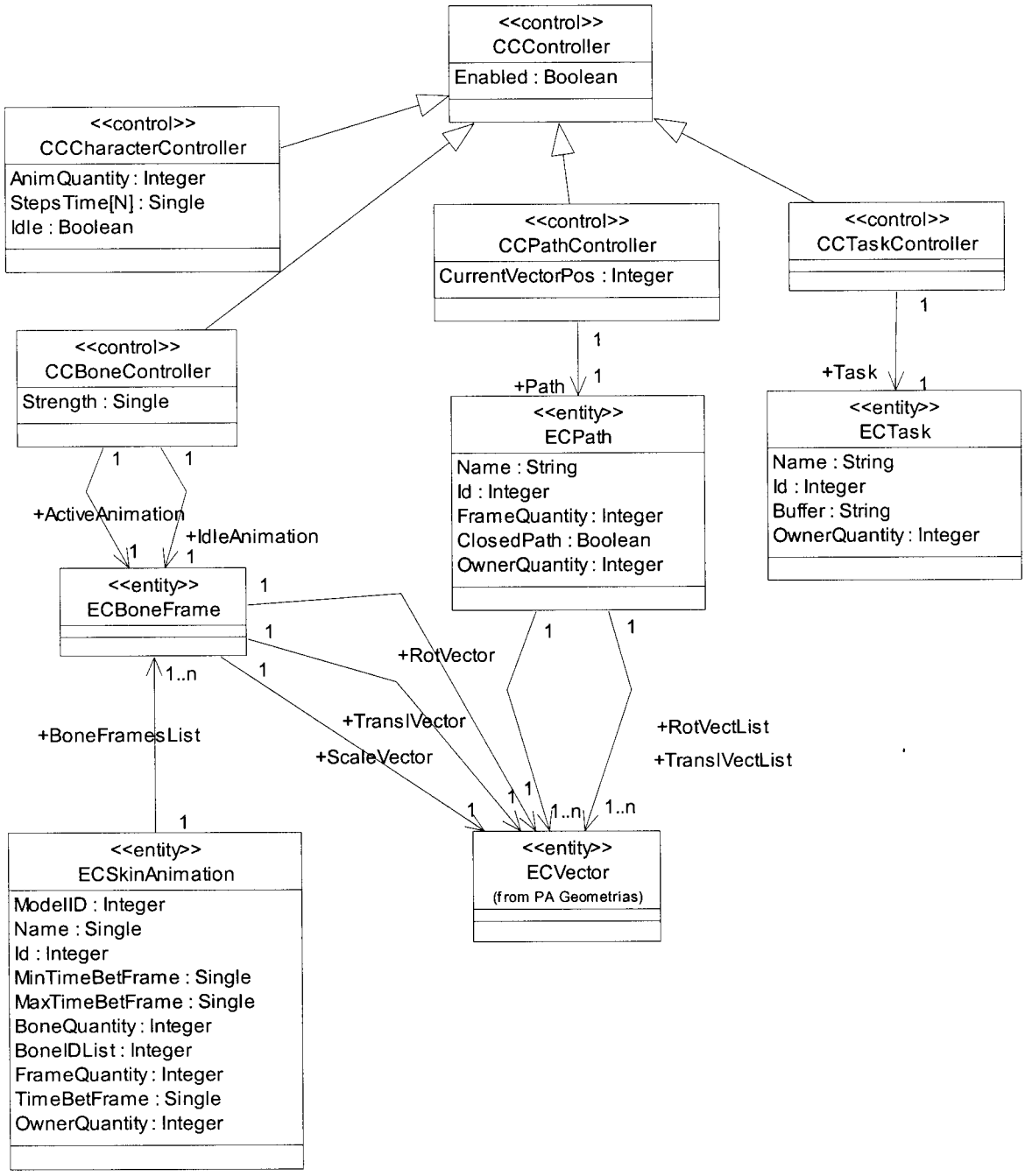

Fig. 96 Diagrama de Clases de Análisis "Manejo de animaciones".

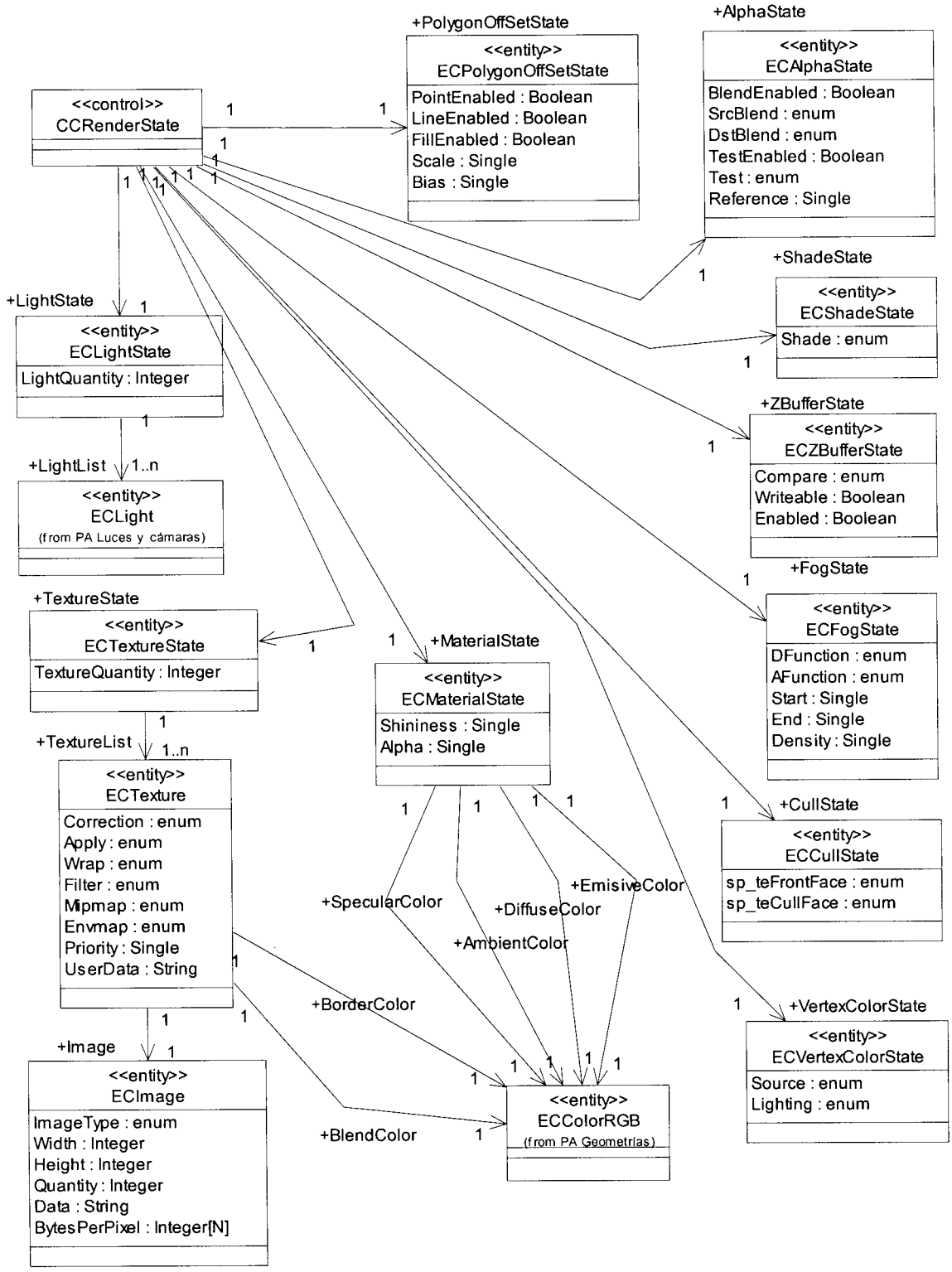

Fig. 97 Diagrama de Clases de Análisis "Estados de Render".

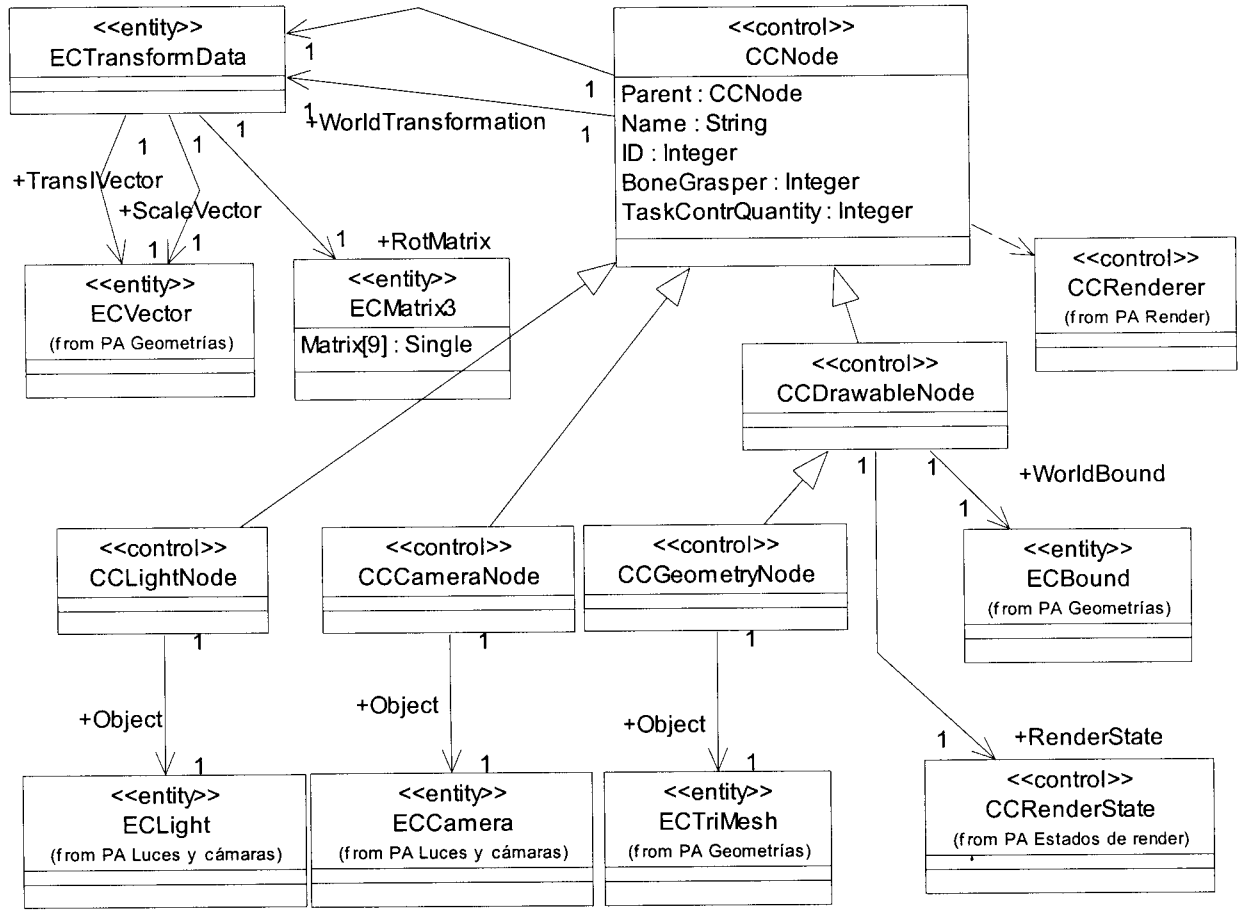

Fig. 98 Diagrama de Clases de Análisis "Manejo del grafo de la escena (A)".

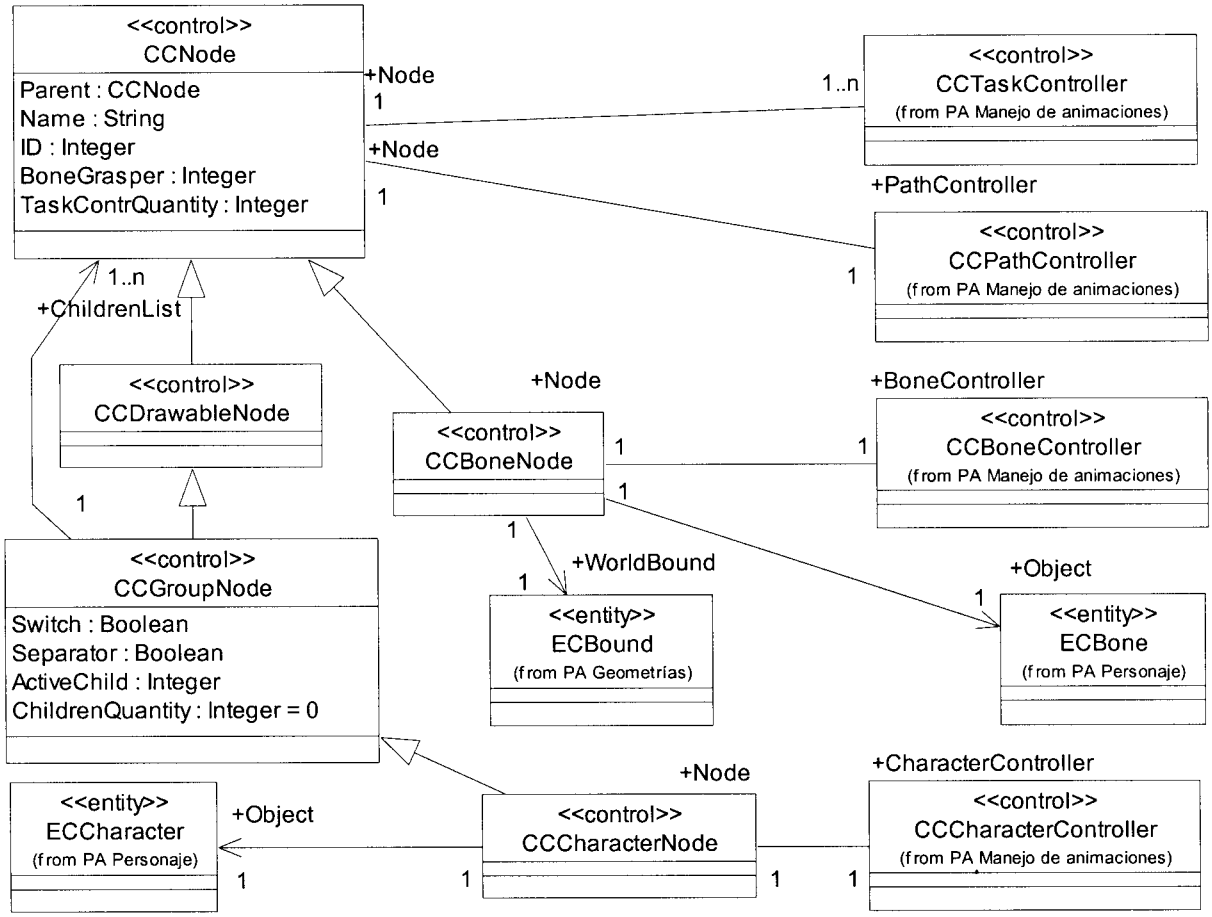

Fig. 99 Diagrama de Clases de Análisis "Manejo del grafo de la escena (B)".

Índice de figuras y tablas

Índice de figuras

Índice de tablas