UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 5 REALIDAD VIRTUAL

SISTEMA DE GENERACIÒN DE FICHEROS PARA ENTORNOS VIRTUALES

Trabajo para optar por el Título de Ingeniero en Ciencias Informáticas

Autores: Wendy García López

Alexis Echemendía González

Tutor: Ing. Fernando Jiménez López

Ciudad de la Habana

Julio, 2007

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Autores:

Wendy García López

_____

Alexis Echemendía González

_____

Tutor:

Ing. Fernando Jiménez López

_____

Dedicatoria

A mis padres, a mis abuelos, a abuela Nena

A mis hermanos, a toda mi familia

Wendy

A mi mamá y mi papá, a mis hermanos.

A toda mi familia.

Alexis

Agradecimientos

Muchas personas han intervenido de diversas formas en la realización de este trabajo, por este medio queremos hacerles llegar nuestros más sinceros agradecimientos, en especial a Fernando que siempre nos apoyó y supo guiarnos por el camino correcto, a María del Pilar por su dedicación desinteresada y su gran ayuda. A la revolución y a nuestro Comandante por darnos la oportunidad de estudiar en la universidad del futuro.

A mi mamá por su amor incondicional, sin su apoyo y comprensión nada de esto hubiese sido posible, a mi papá que fue siempre un ejemplo para mí, por sus consejos y ayuda en todo momento. A mi abuela Hilda que siempre fue como una madre para mí, por su amor y desvelo de toda la vida, a quien le debo en gran medida lo que he logrado hasta hoy, a mi abuelo Enrique por su apoyo incondicional y su gran cariño, a papi (*Pablo*) por estar siempre dispuesto a ayudarme, a mi abuela Alicia por sus consejos y por estar siempre pendiente de mis resultados, a mi abuela Nena que siempre estará presente. A mis hermanitos por quienes trato de sacar lo mejor de mí, para que tengan siempre un buen ejemplo a seguir, a toda mi familia por sus consejos y apoyo, a mis amigos Mileidi y Ernesto que siempre han estado dispuestos a ayudarme.

Wendy

A mi mamá por todo el apoyo, cariño y preocupación, pues sin ella no se hubiese cumplido mi sueño, a mi papá que siempre me aconsejó y me ayudó para que me formara como un buen profesional, a mis hermanos por quienes me esfuerzo para que tengan un buen ejemplo, a tío Chichi y tía Encarna por su cariño, a mis abuelos, a mis primos Lietsi, Mailin, Eney, Juanca y Rodolfito por estar siempre pendiente de mis resultados, a toda mi familia. A todos mis amigos, en especial a Ledian, Pedro José, Evelio y Yismany que estuvieron a mi lado desde mi niñez y a los que conocí en la universidad como Lorenzo (el intruso) y el Jerry.

Alexis

Resumen

Desde el comienzo del estudio de la realidad virtual, se han utilizado diferentes tipos de formatos de ficheros para almacenar el contenido de un objeto 3d o de un entorno virtual complejo; cada persona o entidad que ha decidido trabajar en este perfil ha creado su propio formato para ajustarlo a sus necesidades. El objetivo de este trabajo consiste en elaborar una funcionalidad para exportar escenas u objetos 3d en un formato de ficheros propio.

Para cumplir con los objetivos de esta investigación se hizo un análisis de las diferentes características de los formatos de ficheros gráficos 3d más usados en la actualidad, así como las principales ventajas y desventajas que presentan, además se hace un estudio de las diferentes vías que existen para crear una funcionalidad que exporte un fichero 3d desde el 3d Studio Max.

Como resultado se obtuvo un formato de fichero que optimiza el espacio en memoria, disminuye el tiempo de carga y cumple con las necesidades existentes en el proyecto *Herramientas de Desarrollo para Sistemas de Realidad Virtual* de la facultad 5.

Palabras claves

Formato, fichero, exportar.

Índice de contenido

Introducción

La tecnología ha progresado más rápido que nuestra habilidad para siquiera imaginar que vamos a hacer con ella. Hoy, un proceso digno de la mejor literatura de ciencia ficción, ha trastocado nuestra percepción y está revolucionando el mundo, no sólo de la informática sino también una diversidad de áreas como la medicina, la arquitectura, la educación y la ingeniería entre otras.

La realidad virtual entra en un exclusivo rango de ***herramientas*** *para hacer*, en el cual el usuario puede incursionar creativamente, hasta donde el límite de su imaginación se lo permita. Allí radica, posiblemente el mayor atractivo, por cuanto la imaginación y la creatividad tienen la oportunidad de ejecutarse en un mundo artificial e ilimitado. [9]

La realidad virtual es una representación de las cosas a través de medios electrónicos, que nos da la sensación de estar en una situación real en la que podemos interactuar con lo que nos rodea. [10]

Básicamente consiste en simular todas las posibles percepciones de una persona, como los gráficos para la vista, sonido, tacto e incluso sensaciones de aceleración o movimiento. Todas estas sensaciones diferentes deben ser presentadas al usuario de forma que se sienta inmerso en el universo generado por el ordenador, hasta el punto de dejar de percibir la realidad y ser engañado, sentirse transportado (al otro lado de la pantalla) como si de un universo nuevo se tratase. [3]

Entre los hechos históricos más relevantes se encuentran:

- A finales de los 70 se comienza el estudio de los sistemas de realidad virtual como material para una clase de aviación en el Departamento de Defensa

de los Estados Unidos, para hacer simulaciones de vuelo, practicando y no arriesgando vidas.

- 1980: La compañía StereoGraphics hace las gafas de visión estéreo.
- 1982: Thomas Zimmerman patenta un Electroguante que inventó mientras investigaba sobre cómo controlar con la mano un instrumento musical virtual.
- 1987: La compañía inglesa Dimensión Internacional desarrolla un software de construcción de mundos tridimensionales sobre PC.
- 1988: Scott Foster inventa un dispositivo para la generación de sonido tridimensional.
- 1989: ATARI saca al mercado la primera máquina de galería de vídeo juegos con tecnología 3D. En ese mismo año Autodesk presenta su primer sistema de realidad virtual para PC. [3]

Entre sus aplicaciones se encuentra el entrenamiento militar ya sea de pilotos o soldados, en simuladores de vuelo y tiro respectivamente, en el entrenamiento médico para el aprendizaje y la práctica de forma segura y sin ningún riesgo, en la industria química, con el diseño de drogas, pesticidas y nuevas medicinas, en la industria automotriz para la creación de nuevos modelos y prueba de prototipos, así como en la industria del entretenimiento y muchos más. Todo esto conlleva a obtener grandes beneficios tales como:

- Permite recrear situaciones peligrosas.
- Apoya entrenamiento técnico.
- Facilita la enseñanza de conceptos abstractos (refinación).
- Fácil manipulación y análisis de estructuras complejas.
- Permite explorar diferentes posibilidades para la creatividad y el arte.
- Los estudiantes reciben información rica en estímulos, proveniente de una computadora, utilizando varios sentidos a la vez.

- Adaptación y rehúso de información en realidad virtual a través de objetos de aprendizaje. [4]

En Cuba, poco a poco ha ido creciendo el uso de la realidad virtual. La Universidad de las Ciencias Informáticas (UCI) se ha incorporado a la investigación y desarrollo de este tema con la realización de diversos proyectos, específicamente la facultad 5 es quien sigue esta línea de trabajo, donde actualmente se labora en varios proyectos entre los cuales se pueden mencionar: simuladores de conducción, desarrollo de juegos, simuladores para la defensa (tiro), simuladores quirúrgicos, entre otros.

Por mucho tiempo a la hora de recrear un entorno virtual uno de los grandes problemas ha sido la forma de generar y cargar archivos gráficos que cumplan con los requisitos necesarios, ya sea para disminuir el tamaño de los archivos o para facilitar la carga y puesta en escena de los mismos. Actualmente el proyecto de *Herramientas de Desarrollo para Sistemas de Realidad Virtual*, no cuenta con un formato de fichero propio, que disminuya el tiempo de carga y el espacio en memoria de entornos virtuales, por lo que esto constituye la situación problémica existente. De aquí surge la siguiente interrogante, constituyendo el **problema científico** a resolver, ¿Como elaborar una herramienta que genere un formato de ficheros que sea propio, flexible y que se ajuste a las necesidades del proyecto Herramientas de Desarrollo para Sistemas de Realidad Virtual de la facultad 5?

Como **objeto de estudio** se trabaja en el análisis de los procesos de generación de ficheros para entornos virtuales y el **campo de acción** se basa en el estudio de los procesos de generación de ficheros gráficos 3d para la puesta en escena de entornos virtuales.

El **objetivo** general que propone este proyecto es elaborar un plugin que permita exportar escenarios virtuales complejos, desde el 3D Studio Max que es

la herramienta con la que se trabaja el diseño 3d en el proyecto de Herramientas de Desarrollo para Sistemas de Realidad Virtual.

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

- Investigar las características de los distintos formatos de ficheros gráficos 3d más usados en la actualidad.

- Recopilación de ventajas y desventajas que ofrecen los ya definidos por el software 3dstudiomax y los creados por importantes empresas que desarrollan productos semejantes a los nuestros como por ejemplo: id Software productora de todas las versiones del famoso juego Quake.

- Análisis de los algoritmos empleados para la generación y lectura de estos sistemas de ficheros.

- Elaboración de soluciones técnicas que permitan alcanzar los objetivos propuestos.

- Diseño e implementación del problema.

El presente trabajo está estructurado de la siguiente manera: resumen, introducción, cuatro capítulos de contenido, conclusiones, recomendaciones, referencias bibliográficas, bibliografía consultada, glosario de abreviaturas, glosario de términos y los apéndices.

A continuación se hace una breve descripción del contenido de los capítulos.

-Capítulo 1 “Fundamentación Teórica”, se hace un análisis de la estructura de los formatos de ficheros gráficos 3d mas usados en la actualidad, recopilando ventajas y desventajas que ofrecen cada uno de ellos. También se analizan las distintas vías que existen para elaborar un plugin a partir del 3d Studio Max.

-Capítulo 2 “Soluciones Técnicas”, como lo dice el nombre presenta las características que tendrá el nuevo sistema.

-Capítulo 3 “Descripción de la solución propuesta”, se hace el levantamiento de requisitos y se analiza con mayor profundidad el objeto de estudio.

-Capítulo 4, “Diseño e implementación del sistema”, se muestran los diagramas de clases de diseño agrupados por paquetes, los diagramas de secuencia, las descripciones de las clases de diseño, así como los diagramas de despliegue y de componentes.

Finalmente se muestra un glosario de términos para ayudar al entendimiento del lenguaje técnico que se ha utilizado, así como las referencias bibliográficas por sí el lector desea ampliar más sus conocimientos en el tema.

1

Capítulo 1 : Fundamentación teórica

Introducción

El desarrollo de entornos virtuales ha venido creciendo en gran medida debido a su amplia área de aplicaciones. Se han ido desarrollando herramientas que permitan recrear entornos virtuales y mas adelante herramientas que permitan a un usuario estar inmerso en los mismos, tales como los simuladores y los juegos virtuales. Todo este espacio ficticio debe estar contenido en ficheros gráficos 3d, para después cargarlo por el software que lo pondrá a disposición de un usuario. En este capítulo se hará un estudio de los ficheros gráficos 3d más usados, analizando las ventajas y desventajas que ofrecen, la organización que presenta, las técnicas que utilizan para exportarlos así como los formatos de almacenado que presentan.

1.1 Características que debe tener un formato de fichero gráfico 3d.

Un fichero gráfico 3d debe ser un contenedor de información de una escena, ya sea de un único objeto o de un mundo virtual complejo, que debe contar con todos los atributos que permitan conocer la estructurada de la malla, como son los vértices y caras, en el caso más básico y además suelen tener un conjunto de características que brindan un mayor nivel de detalle a las escenas, entre los cuales podemos mencionar, los materiales, los colores de vértices, las normales de las caras, los estados de render, entre otros. Otro aspecto importante es el formato de almacenamiento y la organización de los atributos en el fichero, este último influye en la complejidad, tiempo de carga y facilidad de entendimiento del mismo.

1.2 Formatos de almacenamiento más usados y ficheros que lo utilizan.

Un formato de almacenamiento es un conjunto de reglas (algoritmo) que define la manera correcta de almacenar datos en memoria. [5]

En la actualidad se utilizan diversos formatos para el almacenamiento de la información en ficheros, entre los que más se utilizan se encuentran el ASCII y el Binario. A continuación se cita algunos ejemplos de ficheros gráficos que usan estos formatos de almacenamiento:

Formato ASCII

Tipos de ficheros que usan este formato de almacenamiento:

- STL
- OBJ
- ASE
- DIRECT X

Formato Binario

Tipos de ficheros que usan este formato de almacenamiento:

- STL
- MD2
- MD3
- DIRECT X
- 3DS

1.3 Características de los formatos de fichero 3D más usados en forma ASCII.

1.3.1 STL (Standard Tessellation Language)

El formato STL es nativo de **stereolithography CAD** creado por **3d System** en Valencia, California, USA. El formato puede encontrarse en datos ASCII o binarios, aunque es más común en formato binario debido a que el tamaño de los archivos son más pequeños. Es uno de los formatos más básicos en cuanto a estructura, muy fácil de cargar y con poca cantidad de atributos, sólo tomando en cuenta, la geometría del objeto (caras y vértices). [6]

Estructura del Fichero STL

solid "nombre": nombre del objeto.

facet normal **float float float: normales de la cara.**

outer loop**: inicialización de los vértices de la cara.**

vertex **float float float: coordenadas del vértice en "x, y, z".**

vertex **float float float**

vertex **float float float**

endloop**: fin de la inicialización de los vértices.**

endfacet**: fin de las declaraciones de cara.**

endsolid "nombre": nombre del objeto, dirección o cualquier otro comentario. [6]

Lo que está en negrita significa que el contenido puede ser repetido una o varias veces, dependiendo de la cantidad de caras, y lo que está en cursiva son las palabras reservadas que usa.

Ventajas

Presenta una estructura muy sencilla a la hora de cargar un fichero.

Desventajas

Sólo analiza el físico de la malla, conteniendo las características básicas para definir la misma, caras y vértices. En un fichero sólo está contenido un único objeto, en caso específico de exportar más de un objeto en un mismo fichero, se toman todos los objetos como una sola geometría, quitando la posibilidad de trabajar con geometrías por separado.

1.3.2 Wavefront OBJ

El formato de archivo obj es un formato de archivo de objetos 3d estándar creado para el uso con Wavefront's Advanced Visualizar.

Es genial para objetos estáticos. Contiene geometrías poligonales que presentan, puntos, aristas y caras, para definir un objeto o geometrías de objetos de forma libre que contienen curvas y superficies, aunque en lo que más se usa es en objetos poligonales. No necesita ningún tipo de encabezado, aunque es usual poner un comentario al inicio del fichero. [7]

Estructura del fichero

Se analizara el fichero para objetos poligonales que son los más usados en los entornos virtuales.

Un comentario como la versión o el autor del modelo.
v float float float: coordenada del vértice.
…

vn float float float: valor de la normal en el punto.

…

vt u v w: valor de las coordenadas de textura.

…

f int/int/int int/int/int int/int/int : información de las caras.

… [7]

Los tres puntos suspensivos (...) significan que se repite el parámetro de acuerdo a la cantidad de atributos del tipo padre que exista. Lo que aparece en negrita y cursiva son las palabras reservadas que utiliza.

Especificaciones de las caras

La información de la cara esta dada por:

f int/int/int int/int/int int/int/int

El primer int es para el número del vértice, el segundo es para el número de la textura y el tercero para el número de la normal.

- Si no presenta normales de vértices y sí coordenadas de mapeo, sería de la siguiente forma.

f int/int

- Si no presenta coordenadas de mapeo, entonces la cara se representa de la siguiente manera.

f int/ /int [7]

Analizando el siguiente ejemplo, se puede ver la estructura que presenta.

```
# Cuadrado de 2 x 2

mtllib master.mtl

v  0.000000 2.000000 0.000000

v  0.000000 0.000000 0.000000

v  2.000000 0.000000 0.000000

v  2.000000 2.000000 0.000000

vt 0.000000 1.000000 0.000000

vt 0.000000 0.000000 0.000000

vt 1.000000 0.000000 0.000000

vt 1.000000 1.000000 0.000000

# 4 vértices

usemtl wood

f 1/1 2/2 3/3 4/4
```

Ventajas

Fácil manejo a la hora de cargar el fichero ya que presenta una estructura sencilla la cual permite guardar toda la información en listas de datos. Un fichero puede contener muchos objetos y tratar a cada uno por separado.

Desventajas

Su uso se centra más en el trabajo con objetos estáticos, ya que no exporta características importantes tales como la matriz de transformación, el bounding box, los colores de vértice, entre otros.

1.3.3 ASE (ASCII Scene Exporter)

Es el preferido por todos los que se inician en la programación de espacios virtuales, para la carga de objetos estáticos.

El formato es basado en un identificador que siempre comienza por el carácter *, ejemplo: *Nombre_del_atributo

Estructura del fichero

El fichero está organizado en forma de jerarquía de clases, las cuales comienzan por un identificador y luego utilizan los caracteres {} para encerrar todo el contenido respecto a este identificador. A continuación se muestra la estructura. [8]

*3DSMAX_ASCIIEXPORT: versión del fichero.

*COMMENT "un comentario"

*SCENE {}

*MATERIAL_LIST {

    *MATERIAL_COUNT: cantidad de materiales.

    *MATERIAL 0 {}

}

*GEOMOBJECT {

    *NODE_NAME "nombre del objeto"

    *NODE_TM {}

*MESH {}

*MESH_FACE_LIST {}

*MESH_NUMTVERTEX: cantidad de vértices.

*MESH_TVERTLIST {}

*MESH_NUMTVFACES: cantidad de caras con textura.

*MESH_TFACELIST {}

*MESH_NUMCVERTEX: cantidad de vértices con colores.

*MESH_CVERTLIST {}

*MESH_NUMCVFACES: cantidad de caras con colores de vértices.

*MESH_CFACELIST {}

*MESH_NORMALS {}

}

*PROP_MOTIONBLUR 0

*PROP_CASTSHADOW 1

*PROP_RECVSHADOW 1

*MATERIAL_REF 0

} [8]

Especificaciones

*SCENE {}: contiene todo respecto a una escena, cuadro en que comienza la animación, cuadro en que termina, velocidad de la animación, color del ambiente, etcétera.

*MATERIAL_LIST {}: contiene información de todos los materiales usados por la malla.

*GEOMOBJECT {}: contiene información de la geometría del objeto.

*LIGHTOBJECT y *CAMERAOBJECT: contiene información acerca de las luces y las cámaras en la escena. [8]

Este es el orden que presenta la estructura del fichero .ASE, analizaremos los identificadores más interesantes.

Identificador de listado de materiales (*MATERIAL_LIST**)**

Estructura del material **(***MATERIAL**)**

*MATERIAL_NAME: Nombre del material.

Estructura del mapa **(***MAP_DIFFUSE**)**

Aquí es donde van los atributos de la textura que se le asignó al objeto.

*BITMAP: Dirección del mapa asignado al objeto. [8]

Identificadores de la geometría (*GEOMOBJECT**)**

*NODE_TM {}: contiene información del bounding box de la malla.

*MESH {}: contiene los atributos que se encuentran debajo.

*MESH_NUMVERTEX: cantidad de vértices.

*MESH_NUMFACES: cantidad de caras. [8]

Estructura del listado de vértices.

*MESH_VERTEX_LIST {

*MESH_VERTEX int float float float

}

El int es para representar el número de vértices y los float son las coordenadas de los vértices “xyz” respectivamente. [8]

Estructura del listado de caras.

*MESH_TFACELIST {

*MESH_FACE int: A: int B: int C: int AB: int BC: int CA: int

El primer entero es el índice de la cara, y los demás son los índices de los vértices para cada esquina del triángulo, y por último AB, BC, BA que se desecha ya que no es necesario.

*MESH_SMOOTHING: grupo de suavizado al que pertenece.

*MESH_MTLID: determina el submaterial al que pertenece. [8]

Estructura de las coordenadas de vértices (textura)

*MESH_NUMTVERTEX: cantidad de vértices con textura.
*MESH_TVERTLIST {
*MESH_TVERT int float float float

El int es el número del vértice y los float son las coordenadas de la textura respecto a los ejes x, y, z del mismo. [8]

Estructura de las coordenadas de caras (textura)

*MESH_NUMTVFACES: cantidad de caras con textura.
*MESH_TFACELIST {
*MESH_TFACE int int int int

El primer int es el número de la cara y los demás son los vértices que definen esa cara tomando el valor de las coordenadas de texturas de los mismos. [8]

Estructura de los colores de vértices.

*MESH_NUMCVERTEX: cantidad de vértices con color.
*MESH_CVERTLIST {
*MESH_VERTCOL int r g b

El int define el número del los vértices y “rgb” definen el color de los vértices (r--> rojo, g--> verde, b--> azul). [8]

Estructura de los colores de caras.
*MESH_NUMCVFACES: número de caras con color.
*MESH_CFACELIST {
*MESH_CFACE int int int int

El primer int es número de la cara y los demás son los vértices que conforman la cara, tomando de ellos sus colores. [8]

Estructura de las normales de las caras
*MESH_NORMALS {
*MESH_FACENORMAL int float float float: normal de la cara.
*MESH_VERTEXNORMAL int float float float: normal del vértice.

Los primeros int son los números respectivos de normal de cara y normal de vértices, los float respectivos a la normal de la cara definen cómo voy a normalizar los vértices respectivos a dicha cara, los demás float representan los valores de la normal en cada coordenada de ejes x, y, z. [8]

Ventajas

Excelente organización de los atributos de un objeto, todo manejado en forma de clases. Contiene todos los atributos necesarios para conformar un buen fichero, tales como; cantidad de materiales, colores de vértices, normales, grupos de suavizado, entre otros. En un fichero contiene más de una malla, por lo que se puede exportar en uno solo escenas completas.

Desventajas

El tamaño del fichero es muy grande.

1.3.4 DIRECT X

El DirectX fue construido para Direct3D y expandido para DirectX 6.0, a partir de ahí ha ido evolucionando debido a su capacidad de expandirse.

Estructura del fichero

El formato está basado completamente a base de plantillas que definen cómo almacenar la información en el fichero. Presenta una estructura jerárquica que permite insertar más plantillas y así expandir el fichero con nuevos bloques de información. [2.1]

A continuación se muestra la estructura de los bloques.

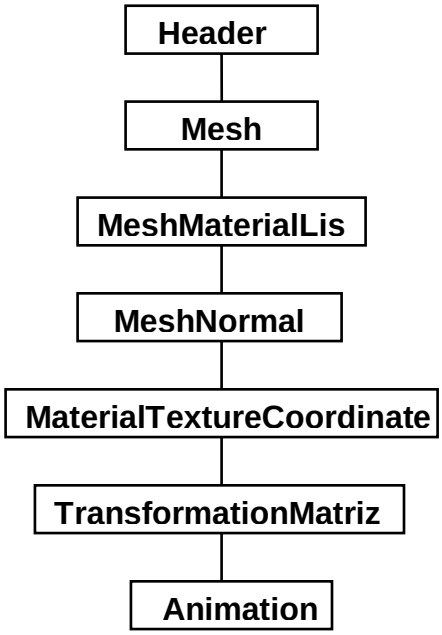

Fig 1: Estructura del directX.

Descripción de los bloques.

A continuación se hace una breve descripción de los bloques más significativos de este formato de fichero que muestra la organización de la estructura interna del fichero.

Header

Contiene información acerca de la versión del exportador, el formato de almacenamiento y el tamaño del fichero en byte.

xof 0302txt 0064: ejemplo de una cabecera, en ella están definidos en qué formato de almacenamiento está el fichero ASCII o Binario. [2.1]

Mesh

Define cómo está estructurada la malla, brindando atributos como son los vértices y las caras.

Mesh Ejemplo

{

int: número de vértices.

float float float: un vértice

…

int: cantidad de triángulos.

int; int, int, int; : información de una cara, el primer int define la cantidad de vértices que la conforman, y los otros tres sus índices.

…

} [2.1]

MeshMaterialList

Define los materiales que se usan en la escena y la relación que existe con los vértices y caras de las geometrías.

MeshMaterialList

{

int: cantidad de materiales que usa la malla.

int: cantidad de caras que usan el material.

int: comienza a listarse el índice de la cara, por cada índice se determina qué material usa, en este caso, para la cara 0 usa el material definido por el int.

…

Material {material1}: nombre del material que usa la malla, la definición se hace fuera del bloque MeshMaterialList. Se definen tantos subbloque Material como cantidad de materiales tenga el bloque MeshMaterialList.

} [2.1]

MeshNormal

Permite definir cómo inciden los rayos de luz en las geometrías.

MeshNormal

{

int: cantidad de vértices.

int int int : valores de la normal de un vértice.

…

int: cantidad de caras.

int; int, int, int;: información de una cara, el primer int define la cantidad de vértices que la conforman, y los otros tres sus índices.

…

} [2.1]

Texturas

Están definidos los caminos de las texturas y las coordenadas de mapeo para cada vértice y cara.

MeshTextureCoordinates

{

int: cantidad de vértices con textura.

int int: coordenadas UV.

…

} [2.1]

Ventajas

Presenta una estructura jerárquica con buena organización y permite ampliar la cantidad de atributos de una escena mediante el uso de plantillas. Optimiza espacio en memoria ya sea en formato ASCII como en binario, aunque la última versión es más óptima.

Desventajas

No cumple con todas las necesidades que se requieren en la herramienta, aunque es muy interesante su estructura y presenta características que serán utilizadas en el formato de la solución propuesta.

1.4 Características de los formatos gráficos 3d más usados en forma Binario.

1.4.1 STL (Standard Tessellation Language)

En forma binaria usa la representación numérica de puntos enteros y flotantes de IEEE. [6]

Estructura del fichero

- string Comentario: un comentario al inicio de fichero.
- unsineg long int: número de la cara.
- float i: normal en i.
- float j: normal en j.
- float k: normal en k.
- **float x**: coordenada para el primer vértice respecto al eje x.
- **float y**: coordenada para el primer vértice respecto al eje y.
- **float z**: coordenada para el primer vértice respecto al eje z.
- **float x**: coordenada para el segundo vértice respecto al eje x.
- **float y**: coordenada para el segundo vértice respecto al eje y.
- **float z**: coordenada para el segundo vértice respecto al eje z.
- **float x**: coordenada para el tercer vértice respecto al eje x.
- **float y**: coordenada para el tercer vértice respecto al eje y.
- **float z**: coordenada para el tercer vértice respecto al eje z.
- **unsined int c**: cuando está en cero indica que terminó la información de la cara.

Lo que está en negrita se repite de acuerdo a la cantidad de caras que tenga el objeto. Los dos últimos bytes son para especificar que aquí termina la información de una cara. [6]

1.4.2 MD2

Es un formato creado por *ID software, inc* para el juego Quake II. Desde el momento que surgió se convirtió en un formato muy popular debido a su facilidad de uso. Los modelos MD2 pueden ser usados para cualquier objeto, tales como: armas, personajes y piezas del terreno. La estructura que presenta es la que se muestra en la siguiente figura. [1.1]

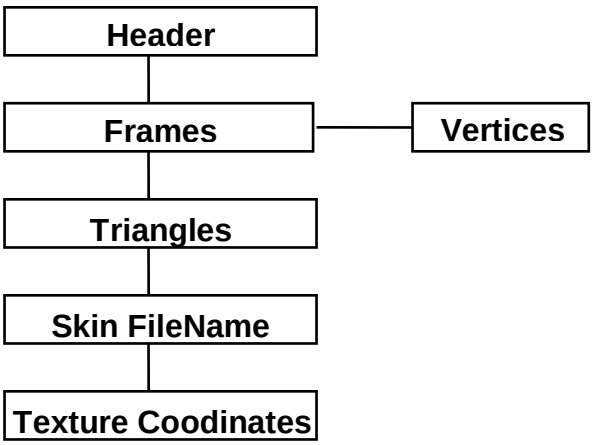

Fig 2: Estructura del formato MD2

Descripción de los bloques

Header: El bloque header contiene informaciones generales, tales como posiciones de un tipo de información en el fichero o cantidad de atributos que contiene un bloque, entre otras. Entre las más importantes se encuentran las que siguen. [1.1]

Los parámetros que contienen este bloque son:

- int m_iVersion: versión del formato.
- int m_iSkinWidthPx: tamaño en cuanto al ancho en pixeles.
- int m_iSkinHeigthPx: tamaño en cuando al alto en pixeles.
- int m_iNumSkin: número de mapas que usa.
- int m_iNumVertex: número de vértices.

- int m_iNumTexturesCoords: número de vértices con textura.
- int m_iNumTriangles: número de triángulos.
- int m_iNumFrames: número de frames.

El propósito de los parámetros que siguen es el de saber en qué lugar del fichero encontrar un tipo de información (offset).

- int m_iOffsetSkin: Skin Filenames.
- int m_iOffsetTexturesCoords: Coordenadas de textura.
- int m_iOffsetTringles: Triángles.
- int m_iOffsetFrames: Frames.

- int m_iSize: tamaño del fichero. [1.1]

Frames y Vértices: Sólo se analizará información referente a los vértices ya que el estudio de este proyecto se basa en el análisis de objetos estáticos.

Los parámetros que contiene este bloque para el caso de los vértices son:

- float m_fScale [3]: matriz de escala.
- float m_fTrans [3]: matriz de translación.
- SMD2Vert m_pVert[iNumVértices]: lista de estructuras vértices que contiene los datos correspondientes a los tres ejes x, y, z. [1.1]

Triangles: Contiene una lista de 3 índices por cada triángulo.

Los parámetros son los que siguen:

- SMD2Tri m_sVertexIndices[m_iNumTriangles]: lista de estructuras triángulos, que presentan cada una tres índices de los vértices que le corresponden.
- SMD2Tri m_sTexIndices[m_iNumTexturesCoords]: listado de estructuras triángulos que definen las coordenadas de texturas en los triángulos de un objeto. [1.1]

Skin Filename: contiene almacenado la raíz de almacenamiento de las texturas que usa el fichero. [1.1]

Textures Coodinates: contiene las coordenadas de mapeo u y v para cada vértices.

Los parámetros son los que siguen:

- SMD2TextCoord m_fTex[m_iTextureCoords]: lista de estructuras que representa las coordenadas de texturas *u* y *v* de los vértices. [1.1]

Ventajas

Al presentar una estructura en bloques, permite que se pueda guardar de manera sencilla, todo el contenido en estructuras de datos. El uso de cabeceras ofrece la ventaja de saber en qué lugar y qué cantidad de un tipo determinado de información contiene el fichero.

Desventajas

Contiene poca información de una escena, le faltan atributos que pueden ser significativos como los colores de vértices, propiedades de los materiales, etc.

1.4.3 MD3

Es un formato creado por *Id Software, Inc* para el famoso juego *Quake III Arena* y otros derivados como: *Retorno al Castillo Wolfenstein*, *Jedi el Caballero Medieval II* y otros. Todos comparten las características de la rápida puesta en escena de los elementos 3d, por lo que está presente la alta optimización de recursos. A continuación se muestra la estructura que presenta. [1.4]

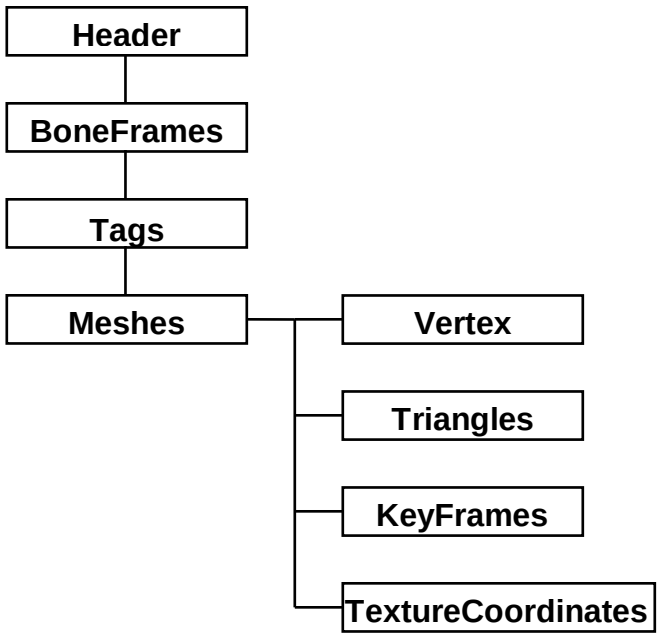

Fig 3: Estructura del fichero MD3

Descripción de los bloques

Header: Cabecera de datos, contiene un id e información de varios bloques del fichero como son cantidad de geometrías, cantidad de conectores, etc, todo para saber dónde se encuentra la información a la hora de cargarla.

Los parámetros que contienen este bloque son los que siguen.

- int m_iID: identificador del fichero.
- int m_iVersion: versión del formato.
- char m_iFilename[32]: nombre del fichero.
- int m_iNumFrames: número de cuadros con animación.

- int m_iNumTags: número de conectores.
- int m_iNumMeshes: número de mallas en la escena.
- int m_iMaxSkin: número máximo de texturas que usa el modelo.
- int m_iHeaderSize: tamaño del encabezado.
- int m_iTagOffset: ubicación de los conectores en el fichero.
- int m_iMeshOffset: ubicación de las mallas en el fichero.
- int m_iFileSize: tamaño del fichero. [1.4]

BoneFrame: Cuadros de huesos, cada uno es un total de 56 bytes que contiene encerrado información del bounding box de la malla por cada fotograma de animación.[1.4]

Tags: Conectores, se usan para conectar en un modelo varias partes, por ejemplo en caso de un personaje, las piernas al torso. Se usa conjuntamente para la animación de los personajes. [1.4]

Meshes: El bloque de las mallas contiene todo lo que se visualiza en la pantalla, los vértices, las caras, las coordenadas de textura y mucho más que en el antiguo MD2.

La estructura que presenta este bloque es la que se muestra a continuación.

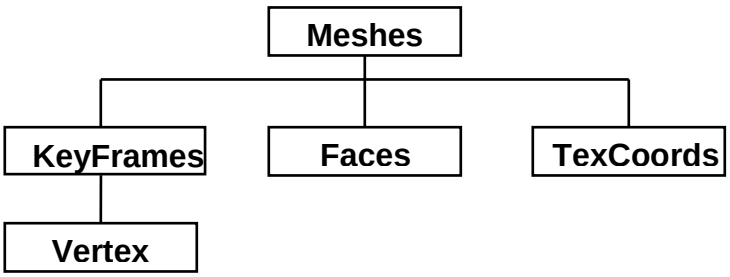

Fig 4: Estructura de la malla

Los parámetros que contienen este bloque son los que siguen:

- char m_cMeshID[4]: identificador de la malla.
- char m_cName[68]: nombre de la malla.
- int m_iNumMeshFrames: número de cuadros con animación en la malla.
- int m_iNumSkin: número de texturas que tiene la malla.
- int m_iNumVerts: número de vértices.
- int m_iNumTriangles: número de caras.
- int m_iTriOffset: ubicación de los triángulos en el fichero.
- int m_iUVOffset: ubicación de las coordenadas de mapeo en el fichero.
- int m_iVertexOffset: ubicación de los vértices.
- int m_iMeshSize: tamaño de la malla en bytes.
- SMd3Faces m_pFaces[m_iNumTriangles]: lista de estruturas de caras donde cada estructura contiene los índices de 3 vértices que conforman la misma.
- SMd3TexCoord m_pTexCoords[m_iNumVerts]: lista de coordenadas de textura, por cada estructura se guardan los valores *u* y *v*.
- SMd3Skin m_pSkins[m_iNumSkin]: lista de directorios de texturas, por cada estructura se guarda una dirección, para ello se reserva memoria para 68 caracteres. [1.4]

Ventajas

Un archivo puede contener más de un objeto. Presenta una estructura en bloques que permite guardar todos los atributos en estructuras de datos de forma organizada según el orden en que se establecen los bloques.

Desventajas

Siguen arrastrando la falta de parámetros significativos desde el MD2, entre los que podemos mencionar los colores de vértices, las propiedades de materiales, entre otros.

1.4.4 3DS

Creado originalmente para 3d StudioMax por Autodesk. Se encuentra en forma de binario y presenta una estructura muy completa. [1.2]

Estructura del fichero

Todos los ficheros están conformados por bloques y casi todos presentan una cabecera que indica donde localizar estos bloques y en qué orden se encuentran, pero el fichero 3ds no muestra esta característica, el formato 3ds puede tener estos bloques en cualquier orden y no presenta una cabecera que especifique donde encontrarlos. Veamos el siguiente ejemplo. [1.2]

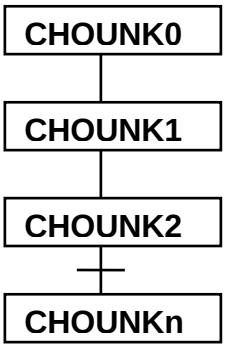

Fig 5: Estructura del 3DS

Chunk que significa bloque de datos.

El fichero 3ds no sólo está dividido en pequeños bloques, sino que cada bloque está dividido en sub-bloques y lo mejor de todo es que no tiene que existir un orden en el fichero, por ello es que a simple vista es muy difícil entender su estructura. El formato es como se muestra a continuación. [1.2]

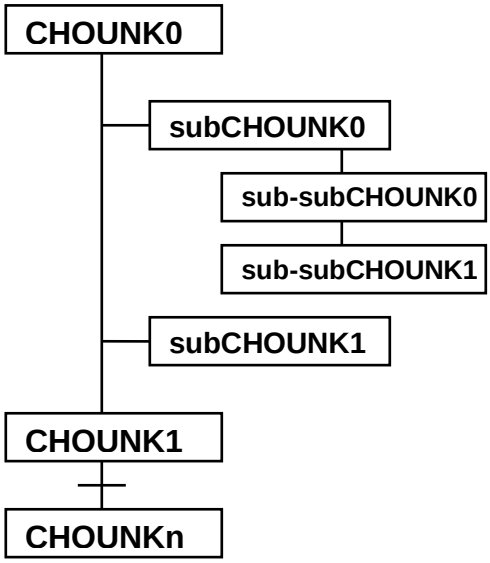

Fig 6: Estructura del 3DS

El fichero presenta una cabecera en cada bloque que contiene un identificador y el tamaño del bloque, entonces dado el tipo de bloque y el tamaño, esto se maneja en el programa que carga el fichero. Un bloque puede contener mallas, luces y cámaras, y los sub-bloques vértices, coordenadas de texturas, etcétera. En caso de que un bloque contenga información que no es de interés, simplemente se salta por encima de él. A continuación mostramos la estructura de la cabecera de un bloque. [1.2]

Header

- unsigned short m_usID: contiene un identificador de 2 bytes que específica qué información contiene este bloque, ejemplo: datos de vértices, entre otros.
- unsigned int m_uiLength: muestra el tamaño del bloque incluyendo la cabecera.

[1.2]

Los atributos son representados por estructuras de datos y cada uno presenta los datos necesarios para los mismos, por ejemplo: la estructura de la cara como se muestra a continuación. [1.2]

Face

- unsigned short m_usIndices[3]: índices de los vértices.
- CVector3 m_vecNormal: normal de la cara. [1.2]

Es muy parecida a la forma en que todos lo ficheros representan una cara, primero por los índices de los vértices y luego la normal de la cara.

El formato 3ds es muy extenso, presenta un gran número de información, por lo cual se hace un análisis general de cómo funciona y qué organización presenta. Para profundizar más en este formato consultar la bibliografía propuesta. [1.2]

Ventajas

Es un formato que optimiza muy bien el tamaño de los ficheros ya que se encuentra en forma binario. Exporta escenas completas, es decir que un mismo fichero puede contener varias mallas.

Desventajas

No muestra un orden en los bloques de datos. La estructura que muestra es compleja y difícil de cargar.

1.5 Software para el diseño de entornos virtuales.

Para la modelación de entornos virtuales se utilizan un gran número de herramientas de diseños, entre las que podemos encontrar, el 3D Studio Max, Maya, XSI, entre otras, lo que posibilita la representación de mundos virtuales que simulen características de realidad. En los proyectos productivos de realidad virtual que se realizan en la facultad cinco se utiliza como software de diseño el 3D Studio Max Versión 5, por su gran facilidad de uso y amplio conocimiento a nivel mundial en el desarrollo de proyectos de este tipo.

1.5.1 Características que presenta la herramienta 3D Studio Max.

El **3D Studio Max** (algunas veces llamado *3ds Max* o simplemente *Max*) es un programa de creación de gráficos y animación 3D desarrollado por Autodesk Media & Entertainment (formalmente conocido como Discreet y Kinetix). Fue desarrollado como sucesor para sistemas operativos Win32 del 3D Studio creado para DOS. Kinetix fue más tarde fusionada con la última adquisición de Autodesk, Discreet Logic. La versión de noviembre del 2006 es 3ds max 9 en inglés y la 8 en español. [11]

Es uno de los programas de animación 3D más utilizados. Dispone de una sólida capacidad de edición, una omnipresente arquitectura de plugins y una larga tradición en plataformas Microsoft Windows. 3ds Max es utilizado en mayor medida por los productores de videojuegos, aunque también en el desarrollo de proyectos de animación como películas o anuncios de televisión, efectos especiales y en arquitectura. [11]

Con más de una década de orientación a los juegos, el software 3ds max de Discreet es utilizado por más del 80% de los principales productores y editores de juegos, por sus funciones específicas, sus productivos paquetes de herramientas y su acceso a la mayor comunidad mundial que más activamente comparte ideas, scripts y otras funcionalidades(plugins). [11]

Posee completas herramientas 3d con múltiples enfoques para abordar la producción a su manera. Optimización integrada con gráficos de alto rendimiento y plataformas específicas. SDK avanzado para un desarrollo personalizado. Foros activos de la Comunidad de Discreet para ayuda con información técnica y cientos de scripts y herramientas gratuitas. [11]

Principales funciones

- Herramientas de modelado para polígonos y parches sin precedentes.

- Completas herramientas para animación de personajes con total IK y Skinning.

- Herramientas de color de vértices basadas en capas.

- Sistema de integrado de partículas orientado a eventos.

- Visión esquemática mejorada desarrollada con la principales compañías de juegos para la gestión de escenas complejas Sombreado Dinámico UI, crea componentes UI amigables para los artistas para Sombreado HLSL, basados en parámetros dentro del archivo DirectX.FX.

- Emulación de datos del motor de juego dentro del puerto de visión de 3ds max.

- Interfaz de exportación de juegos que facilita el envío de datos a motores específicos.

- Incorpora dos caminos para crear nuevas herramientas (plugins) de trabajo que son el maxScript y el SDK. [11]

Lenguajes que brinda para el desarrollo de la funcionalidad.

- MaxScript
- SDK(c++)

1.5.1.1 Características del MaxScript.

En general los plugin creados por MaxScript corren más lentos en comparación con los escritos en C++, al punto de que sí el rendimiento es la cuestión, usar el SDK será lo más sugerente.

Por otro lado MaxScritp brinda algunos métodos de alto nivel que pueden ser encontrados en el SDK(C++) de max y presenta nuevas potencialidades que no están presentes en el SDK. Sí la funcionalidad es soportada por el MaxScritp, pero no por el SDK, entonces el MaxScritp será la única opción. En particular el 3ds Max brinda potencialidades vía el OLE/ActiveX que hace más fácil codificar en MaxScript que con el SDK.

El MaxScript puede ser conveniente para crear prototipos de plugins, para desarrollar plugin de simple complejidad y para escribir pruebas de pequeños fragmentos de funcionalidades. [12]

1.5.1.2 Características del SDK(C++)

El SDK es el preferido cuando el rendimiento es lo más importante; en general cuando el cálculo es el principal propósito del plugin. [12]

Debido al gran impacto del 3ds Max en el mundo de la realidad virtual se ha incorporado una nueva librería al SDK, llamada **IGame.lib**, mediante la cual se puede acceder a todos los atributos de los nodos de una escena con un pequeño esfuerzo en comparación con los métodos existentes anteriormente.[12]

Conclusiones

En el transcurso de este capítulo, como base para el entendimiento del tema en que se desenvolverá este proyeto, se realizó un análisis de los distintos tipos de formatos de ficheros 3d más usados en el mundo de la Realidad Virtual. Además se mostraron las principales características de la herramienta de diseño 3d que se utilizará para exportar el formato de fichero y los lenguajes que brinda para el desarrollo de esta funcionalidad.

Se logró también entender el funcionamiento básico de las estructuras de los distintos formatos de ficheros análizados, y se definieron las principales ventajas y desventajas que presentan cada uno de ellos.

2

Capítulo 2 : Soluciones técnicas

Introducción

En el presente capítulo se proponen soluciones técnicas para la estructura del fichero gráfico 3d y soluciones para su exportación desde el 3ds Max.

2.1 Forma de Salvado.

El fichero se exportará en **binario**, pues como se analizó en la fundamentación teórica, los formatos de ficheros de este tipo son de menor tamaño que los que están en forma de texto plano, ejemplo de esto son el **md3** y el **unreal**, que son propios de dos grandes empresas que desarrollan proyectos de realidad virtual con gran impacto en el mundo de los videojuegos. Además el poseer esta forma de salvado brinda la posibilidad de tener confidencialidad en la información contenida en el fichero 3d.

2.2 Estructura y nombre del fichero.

El nuevo formato se llamará .STK que se debe a la herramienta para la cual se está elaborando, el nombre completo de la herramienta es Scene Tool Kit donde las abreviaturas son STK, de ahí el nombre. La organización está dada en bloques de datos, en la siguiente figura se muestra de forma clara la estructura que presenta.

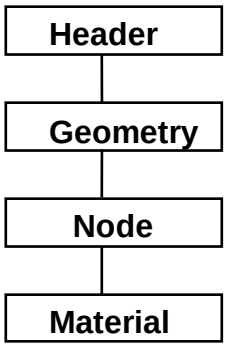

Fig 7: Estructura del fichero

El fichero estará organizado de esta manera, primero el encabezado (Header), después las geometrías (Geometry), los nodos (Node) y finalmente los materiales (Material). En la fundamentación teórica se muestra como la mayoría de los formatos ya existentes usan una organización en forma de bloques, este tipo de estructura permite que a la hora de leer el fichero, se cargue un bloque de datos por completo, el cabezal del disco duro sólo tendrá que moverse en

una sola dirección una vez que se está leyendo un bloque, logrando mayor velocidad de lectura cuando se está cargando el fichero.

2.2.1 Características del bloque Header.

Header es el bloque que representa el encabezado del fichero 3d, brindando informaciones generales tales como la versión del formato, el nombre de la escena, el tamaño del fichero, entre otros. A continuación se muestra la estructura del encabezamiento (**Header**) para un mejor entendimiento.

- unsined int h_iID: identificador que sirve para saber con qué fichero se está trabajando.
- unsined int h_iVersion: se usa para guardar la versión del formato, almacenando la misma en 4 bytes.
- unsined int h_iHeaderSize: tamaño de la cabecera, lo cual proporciona la posibilidad de saber cuándo se termina de leer la cabecera.
- unsined int h_iReserved1: espacio reservado para posible uso en un futuro, permitiendo insertar cualquier tipo de dato de interés.
- unsined int h_iReserved2: nuevo espacio reservado para posible uso en un futuro, posee las mismas características que el anterior.
- unsined int h_iTriMeshQuantity: cantidad de objetos de tipo TriMesh en la escena (mallas poligonales).
- unsined int h_iPolyLineQuantity: cantidad de objetos tipo PolyLinea en la escena (formas).
- unsined int h_iNodeQuantity: cantidad de nodos de una escena.
- unsined int h_iMaterialQuantity: cantidad de materiales que se usan en la escena.
- unsined int h_iTextureQuantity: cantidad de texturas que son usadas por los materiales.
- char h_acTextureDir[32]: nombre de la carpeta que contiene las texturas.

¿El por qué de usar un encabezado?

El encabezado es un contenedor de informaciones generales que describen aspectos básicos del fichero, dando la posibilidad de conocer qué versión de formato es con la que se está trabajando, cuántos elementos de un tipo de bloque están en el fichero, entre otros, que bien pueden incluirse en los espacios reservados que se dejan en la cabecera para un uso futuro.

2.2.2 Características del bloque Geometry

Geometry contiene toda la información respecto a las geometrías presentes en una escena. A continuación se muestra la figura donde se ven los tipos de geometrías que exportará el plugin y la característica que presenta cada una de ellas.

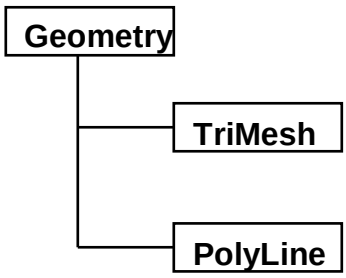

Fig 8: Estructura del bloque geometría

Existen dos tipos de geometrías a exportar, una malla (TriMesh) y una polilínea (Polyline).

Los parámetros que contienen el bloque geometría son comunes para los dos tipos de geometría:

- char g_acName: nombre de la geometría.
- unsined int g_iID: Id de la geometría.
- unsined int g_iVertexQuantity: cantidad de vértices que contiene la geometría.
- sVertex g_TVertexList[g_iVertexQuantity]: arreglo de estructuras vértices dados en los tres ejes *x, y, z*.

Las geometrías se exportarán en ese orden, primero todas las mallas (TriMesh) y después todas las polilíneas (Polyline), logrando así mayor comodidad a la hora de cargar el fichero una vez exportado.

¿El por qué de usar el bloque de geometrías?

El bloque geometría constituye la principal parte de información del fichero, sin el mismo no existiría nada que visualizar una vez exportada la escena, por lo cual es el más importante dentro del fichero. Al tener dentro dos subbloques, brinda mayor comodidad y organización, ordenando primero todas las geometrías tipo malla y después todas las tipo polilínea.

2.2.2.1 Características del subbloque TriMesh.

TriMesh contiene todo referente a una malla, dígase un objeto poligonal. En la siguiente figura se muestra la estructura que presenta con sus respectivos atributos.

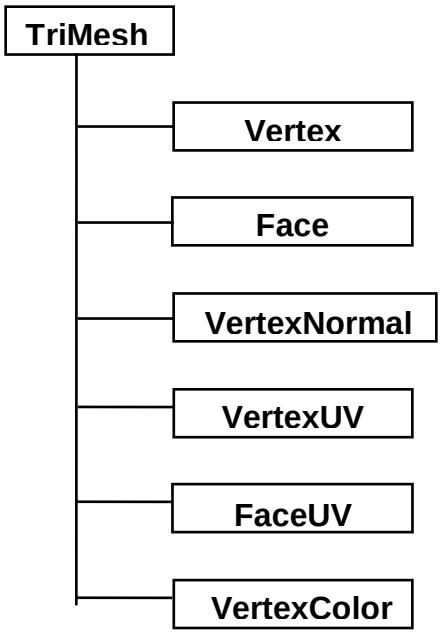

Fig 9: Estructura del subbloque TriMesh

Los parámetros que contienen al subbloque TriMesh son los que siguen a continuación

- unsined int t_TriQuantity: cantidad de triángulos que conforman la malla.
- sTriMesh t_TriList[t_TriQuantity]: lista de estructuras triángulos, cada estructura conformada por tres índices de los vértices correspondiente al triángulo.
- sVertex 3*t_VertexNormalList[t_TriQuantity]: lista de estructuras de normales de vértices dada por las caras.
- bool t_ListExist: byte de información que se usa para saber qué atributos contiene la malla; un bit para cada existencia del atributo, los atributos son los que siguen: t_VertexColorList, t_TriColorList, t_UVList0, t_UVList1, t_UVList2 y t_UVList3.

- unsined int t_VertexTQuantity0: cantidad de vértices con textura en el canal de mapeo1.
- unsined int t_VertexTQuantity1: cantidad de vértices con textura en el canal de mapeo 2.
- unsined int t_VertexTQuantity2: cantidad de vértices con textura en el canal de mapeo 3.
- unsined int t_VertexTQuantity3: cantidad de vértices con textura en el canal de mapeo 4.
- unsined int t_VertexCQuantity: cantidad de vértices con color.
- sVertex t_VertexUVList0[t_VertexQuantity]: lista de coordenadas de textura referente a los valores u y v para el mapa difuso.
- sTriMesh t_TriUVList0[t_TriQuantity]: lista de estructuras de texturas de caras, cada una compuesta por los índices de los vértices con textura correspondientes al canal.
- sVertex t_VertexUVList1[t_VertexTQuantity]: memoria reservada a futuro uso.

- sTriMesh t_TriUVList1[t_TriQuantity]: lista de estructuras de texturas de caras, cada una compuesta por los índices de los vértices con textura correspondientes al canal.
- sVertex t_VertexUVList2[t_VertexTQuantity]: lista de coordenadas de textura para el mapa de luces.
- sTriMesh t_TriUVList2[t_TriQuantity]: lista de estructuras de texturas de caras, cada una compuesta por los índices de los vértices con textura correspondientes al canal.
- sVertex t_VertexUVList3[t_VertexTQuantity]: memoria reservada para futuro uso.
- sTriMesh t_TriUVList3[t_TriQuantity]: lista de estructuras de texturas de caras, cada una compuesta por los índices de los vértices con textura correspondientes al canal.
- sVertexC t_VertexColorList[t_VertexCQuantity]: lista de colores de vértices, dados en la combinación de los tres colores que usa OpenGL para pintar r(rojo), g(verde), b(azul).
- sTriMesh t_TriColorList[t_TriCQuantity]: lista de estructuras colores triángulos, cada estructura está conformada por tres índices de los vértices con color correspondientes al triángulo.

2.2.2.2 Características del subbloque Polyline.

El bloque Polyline contiene todo lo referente a una polilínea en el 3d Studio Max, es decir una línea, conformada por un conjunto de puntos, pudiendo ser abierta o cerrada. En la siguiente figura se muestra la estructura del subbloque Polyline.

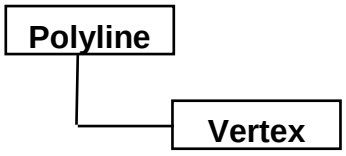

Fig 10: Estructura del subbloque Polyline

Los parámetros que contienen este subbloque son los que siguen:

- bool p_Close: define si es una polilínea cerrada, en caso de no serlo es continua.
- sVertex p_VertexList[p_VertexQuantity]: arreglo de gvértices.

2.2.3 Características del bloque Node.

El bloque Node brinda toda la información respecto a los nodos en una escena. Existen dos tipos de nodos, los nodos geometría y los nodos grupos, en la siguiente figura se muestra de forma clara la estructura de este bloque.

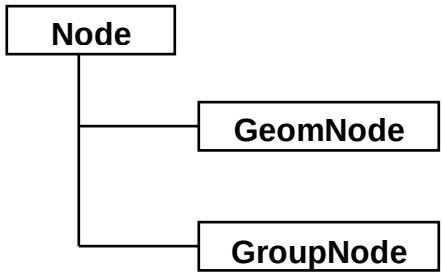

Fig 11: Estructura del bloque Node

¿El por qué de usar el bloque de nodo?

El bloque nodo trata a cada objeto en la escena como un todo, analiza sólo las características generales y propiedades desde el punto de vista de cómo se van a representar en la escena, datos tales como: el grado visibilidad, la matriz de transformación, entre otros. Está dividio en dos subbloques, lo cual presenta gran ventaja, a continuación se analizan las mismas en cada subbloque.

- GeomNode: optimiza el tamaño del fichero ya que en caso de que en la escena existan objetos que sean instancias o copias idénticas de otro objeto, sólo se exporta en el fichero un único objeto y un nodo por cada instancia, los cuales contienen la matriz de transformación del objeto que

representa dónde están ubicados en la escena. Un ejemplo típico es cuándo se exporta un entorno donde hay mucha vegetación, generalmente se crean instancias de un mismo tipo de árbol o arbusto por todo el terreno, por lo que se exportaría un solo árbol y los nodos que son instancias del mismo.

- GroupNode: optimiza el dibujado de la escena y el trabajo con la misma una vez que se cargó el fichero. [13]

A continuación se muestra una estructura más detallada de la representación de los nodos en la escena.

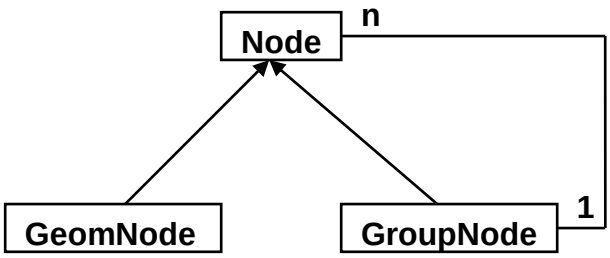

Fig 12: Estructura del subbloque GroupNode

Ambos tipos de nodos contiene los mismos parámetros, sólo que varían de acuerdo al tipo. Los parámetros son los siguientes.

- int nNode_ChildrenQuantity: cantidad de hijos de un nodo, para el caso de un nodo geométrico es 0.
- bool nNode_Children_Brother: un byte que se usa para saber si es hermano o hijo del nodo anterior, sí el primer bit está activado es hermano, sino es hijo.
- int nNode_Index: índice de la geometría que está contenida en el nodo geométrico, en caso de ser un nodo grupo su índice es -1.
- int nNode_MaterialIndex: índice del material que usa el nodo. En el caso particular de que no use ningún material toma valor -1.
- float nNode_TMatriz [4][3]: matriz de transformación que exporta el 3d Studio Max.

- bool nNode_BackFaceCull: en caso de que sea 1 los triángulos mirados por detrás son invisibles, en caso que sea 0 visibles.
- float nNode_Visibility: nivel de visibilidad, está dado entre cero y uno.
- int nNode_UserPropertiesSize: tamaño que ocupa un comentario acerca del nodo dado por el usuario.
- char nNode_UserProperties[ng_UserPropertiesSize]: definición dada por el usuario al nodo.

¿Cómo se exporta la lista de hijos?

Primeramente se contruye el árbol de nodos donde se crea un padre jerárquico que es la cabeza de todos los nodos de la escena, una vez construido se va haciendo un recorrido en preorden y guardando en una lista para luego exportar la misma.

En la siguiente figura se muestra un ejemplo de cómo sería la estructura del árbol.

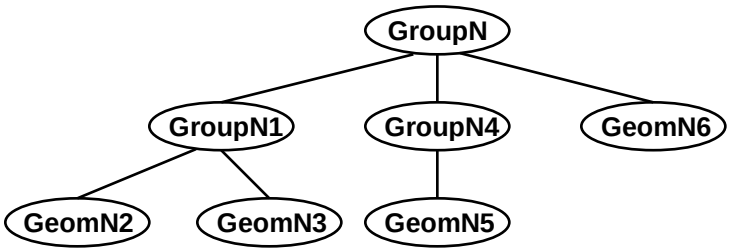

Fig 13: Estructura del árbol de nodos

Nota: El orden en que se exportaría está dado por el número que presenta en la figura anterior.

2.2.4 Características del bloque Material.

El bloque Material contiene todo lo referente a los materiales que se utilizan en la escena y las características de las texturas. A continuación se muestra una figura que representa la estructura de este bloque.

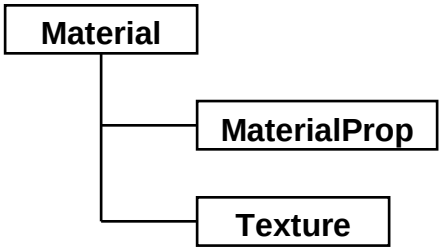

Fig 14: Estructura del bloque Material

¿El por qué de usar el bloque Material?

El objetivo de utilizar este bloque es el de tratar a un material como un objeto con sus respectivas propiedades. Se exportarían todos los materiales que se usan en la escena, en caso de no haber separado este bloque, por cada nodo tendría que dar las características del material que tiene asignado, lo que sería una desventaja ya que en caso de que un nodo use el mismo material que otro, se tendría que repetir la misma información nuevamente. Por otra parte, el tener dividido el bloque material en dos subbloques persigue un objetivo semejante al descrito anteriormente, el primer subbloque contiene las propiedades de los materiales y los índices de las texturas que usan, y el segundo bloque contiene los caminos de las texturas, por lo que se puede tener una cantidad *n* de materiales que utilicen una misma textura sin tener que volver a definirla, sólo haciendo referencia a ella.

2.2.4.1 Estructura del subbloque MaterialProp.

El subbloque MaterialPro contiene todas las propiedades de los materiales en una escena.

Los parámetros que contienen son los que se exponen a continuación.

- char mp_Name[32]: nombre del material.
- bool mp_TwoSide: en caso de ser 1 está activado el modo de dos lados, en caso de ser 0 lo contrario.
- int mp_AmbientColor[2]: color ambiental dado en rgb.
- int mp_DiffuseColor[2]: color difuso dado en rgb.
- int mp_SpecularColor[2]: color especular dado en rgb.
- float mp_Opacity: nivel de opacidad.
- int mp_IndexAmbient: índice de la textura que se le pasó por el canal ambiente, en caso de no tener asignada ninguna toma valor -1.
- int mp_IndexDiffuso: índice de la textura que se le pasó por el canal difuso, en caso de no tener asignada ninguna toma valor -1.
- int mp_IndexOpacity: índice de la textura que se le pasó por el canal de opacidad, en caso de no tener asignada ninguna toma valor -1.
- int mp_IndexSpecular: índice de la textura que se le pasó por el canal especular, en caso de no tener asignada ninguna toma valor -1.
- int mp_IndexMaterial1: reservado.
- int mp_IndexLighMap: índice de la textura que se le pasó por el canal de mapa de luces, en caso de no tener asignada ninguna toma valor -1.
- int mp_IndexMaterial2: reservado.

2.2.4.2 Estructura del subbloque Textura.

El subbloque Textura contiene información referente a las texturas que se usan en una escena, las texturas en este subbloque no son más que los ficheros que representan una imagen dígase un bmp, un tga u otro. Este subbloque contiene el nombre de la textura con su extensión.

Los parámetros que contiene son los siguientes.

- char te_Name[32][m_TextureQuantity]: lista de caminos y nombres de texturas.

2.3 Aspectos generales.

Para exportar el formato se hará uso de la librería IGame contenida en el sdk de Max ya que, como se analizó en la fundamentación teórica, está es una vía que brinda flexibilidad y rapidez a la hora de crear una nueva funcionalidad para 3ds Studio Max. La programación será en lenjuague c++ porque es la que se emplea para desarrollar funciones con el sdk. El diseño e implementación se hará utilizando la filosofía de programación orientada a objetos.

Conclusiones

En este capítulo quedan plasmadas las bases de la estructura del nuevo formato. Contiene los atributos necesarios para desarrollar el plugin, por lo que a partir de éstos se desarrollaran un conjunto de clases que permitan exportar el fichero con las características que se desean y así dar cumplimiento a los objetivos del proyecto.

3

Capítulo 3 : Descripción de la solución propuesta

Introducción

El presente capítulo se enmarca en la conceptualización de la solución propuesta, se trabaja sobre las necesidades que existen en la herramienta para la cual se está desarrollando este trabajo.

3.1 Reglas del negocio

Las reglas de negocio son una serie de restricciones de la organización a la hora de realizar una determinada actividad, e incluyen las restricciones asociadas a las informaciones (restricciones de integridad) y a las actividades, por lo que regulan algún aspecto del negocio, por esta razóon es de gran importancia identificarlas dentro del negocio, evaluar sí son relevantes dentro del campo de acción que se está modelando e implementarlas en la propuesta de solución.

A continuación se definen las reglas para este modelo de dominio, orientadas a los diseñadores gráficos que exporten escenarios virtudes con el nuevo formato de fichero fichero.

- Los ficheros de textura deben ser bmp o tga, en caso de no tener este formato será tomado como un fichero corrupto.
- Para generar el fichero es necesario tener instalado el 3d Studio Max 8, pues para realizar el plugin se utiliza la librería IGame v.2 que sólo trabaja con esta versión del software.
- Tiene que haber algún objeto geométrico en la escena para que pueda ser generado un fichero.
- Una polilínea sólo puede contener una línea, en caso contrario no se exportará.
- Un nodo geométrico sólo puede tener asignado un único material, no se permiten multimateriales.
- Un material puede tener asignado como máximo una textura.

3.2 Modelo del dominio.

A continuación se muestra el diagrama de dominio que servirá como referencia a los posibles usuarios para la comprensión de todos los términos utilizados en la elaboración del plugin y como guía para el futuro diseño del sistema.

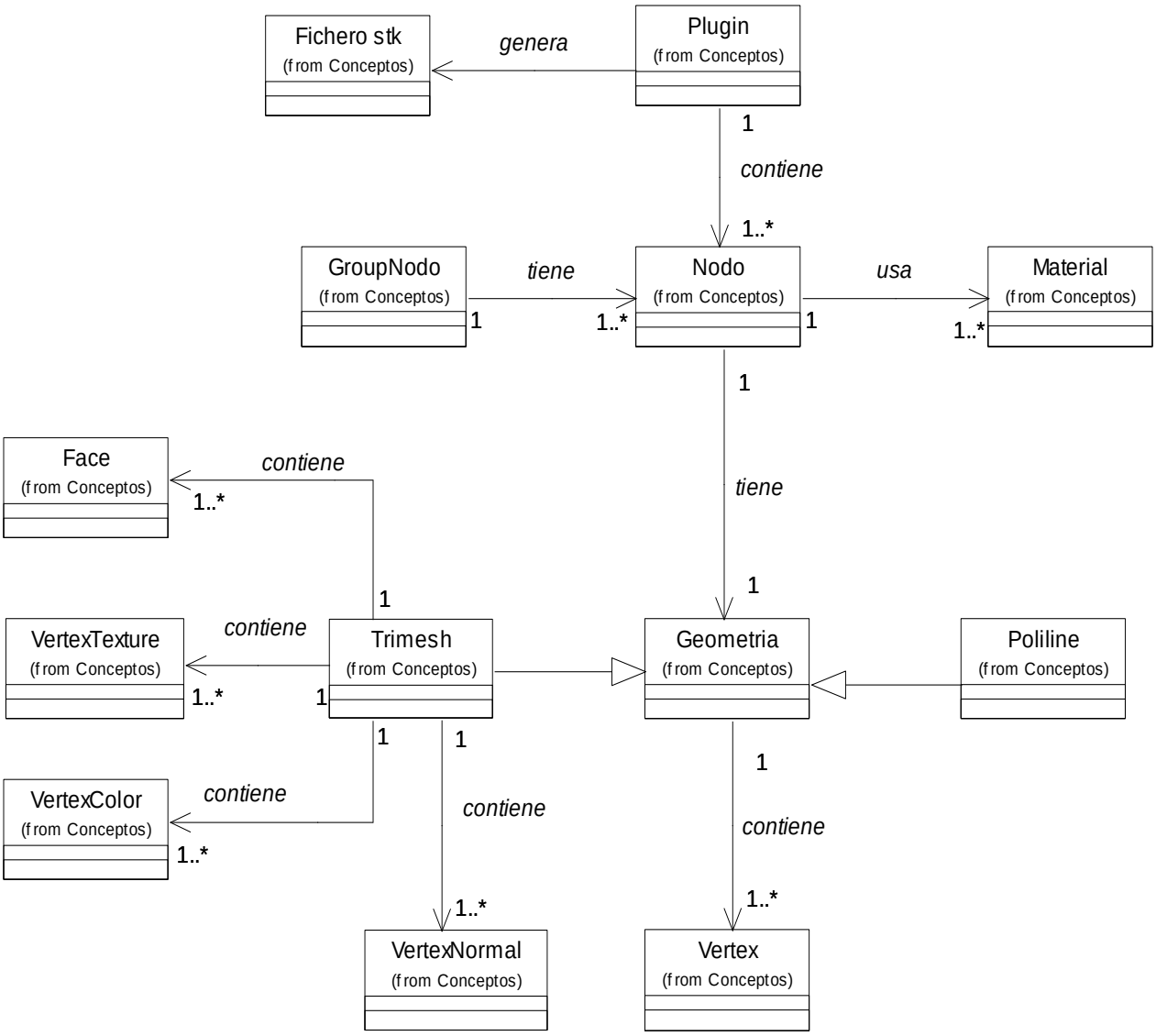

Fig 15: Modelo de Dominio

3.2.1 Glosario de términos del modelo de dominio.

Se hace una breve descripción de los principales conceptos manejados en el modelo de dominio para lograr un mejor entendimiento del mismo.

Plugin: Es un contenedor de una funcionalidad que permiten a un software realizar tareas adicionales.

Nodo: Contiene una estructura geométrica y se encanga de manipularla en un escena a través de atributos como la matriz de transformación y los estados de render.

Material: Contiene información acerca de las imágenes que serán proyectadas en una geometría en diversos canales del material como: difuso, especular, mapa de luces, entre otros.

Geometría: Contiene información del físico de las geometrías, de cómo está estructurada la misma, presenta atributos tales como: vértices, caras, coordenados uv, entre otros. Pueden ser de dos tipos, TriMesh que definen mallas poligonales y PolyLine que define líneas.

VertexNormal: Define cómo incide la luz sobre un vértice.

VertexTexture: Define la posición de un vértice en una imagen o textura.

VerteColor: Define el color de un vértice en RGB (Red, Green and Blue).

Face: Está formada por la unidad mínima de puntos que conforman un plano, los índices de tres vértices.

3.3 Captura de requisitos

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

3.3.1 Requisitos funcionales

1. Identificar visibilidad del nodo.

2. Almacenar lista de geometrías.
   2.1 Identificar tipo de nodo.
   2.2 Identificar instancias de un nodo.
   2.3 Eliminar instancias de un nodo.
   2.4 Extraer cantidad de TriMesh.
   2.5 Extraer cantidad de Polyline.

3. Organizar lista de geometrías.

4. Crear geometría genérica.
   4.1 Extraer nombre de la geometría.
   4.2 Extraer id de la geometría.
   4.3 Extraer cantidad de vértices.
   4.4 Extraer cantidad de caras.
   4.5 Extraer vértice.
   4.6 Extraer vértice de color.
   4.7 Extraer normal de vértice.
   4.8 Extraer vértice de textura.
   4.9 Extraer cara.
   4.10 Extraer canal de mapeo.
   4.11 Almacenar información en la geometría genérica.

5. Exportar geometrías.
   5.1 Exportar cantidad de TriMesh.
   5.2 Exportar cantidad de Polyline.
   5.3 Exportar nombre de la geometría.
   5.4 Exportar id de la geometría.
   5.5 Exportar cantidad de vértices.
   5.6 Exportar cantidad de caras.
   5.7 Exportar vértice.
   5.8 Exportar vértice de color.
   5.9 Exportar normal de vértice.
   5.10 Exportar vértice de textura.
   5.11 Exportar cara.
   5.12 Identificar tipo de polilínea.
   5.13 Exportar tipo de polilínea.

6. Almacenar lista de nodos.
   6.1 Validar tipo de nodo.
   6.2 Actualizar índice de geometría.
   6.3 Actualizar índice del material.

7. Crear nodo genérico.
   7.1 Extraer cantidad de nodos.
   7.2 Extraer cantidad de hijos.
   7.3 Extraer parentesco con el nodo anterior.
   7.4 Extraer índice de geometría.
   7.5 Extraer índice del material.
   7.6 Extraer matriz de transformación.
   7.7 Extraer estados de render.
   7.8 Extraer visibilidad de parte trasera de las caras.
   7.9 Extraer visibilidad de un nodo.

7.10 Extraer tamaño del comentario.

7.11 Extraer comentario.

7.12 Almacenar información en nodo genérico.

8. Exportar nodo.

8.1 Exportar cantidad de nodos.

8.2 Exportar cantidad de hijos.

8.3 Exportar parentesco con el nodo anterior.

8.4 Exportar índice de geometría.

8.5 Exportar índice del material.

8.6 Exportar matriz de transformación.

8.7 Exportar estados de render.

9. Almacenar lista de materiales.

9.1 Identificar tipo de material.

9.2 Almacenar propiedades de un material.

9.3 Identificar formato de textura.

9.4 Eliminar instancias de textura.

9.5 Almacenar camino de textura.

10. Crear material genérico.

10.1 Extraer nombre.

10.2 Identificar textura 2-Caras.

10.3 Extraer color de ambiente.

10.4 Extraer color difuso.

10.5 Extraer color especular.

10.6 Extraer nivel de opacidad.

10.7 Extraer índice de la textura difusa.

10.8 Extraer índice de la textura de opacidad.

10.9 Extraer camino de la textura.

10.10 Almacenar información en material genérico.

11. Exportar material.
    11.1 Exportar nombre.
    11.2 Exportar tipo de textura 2-Caras.
    11.3 Exportar color de ambiente.
    11.4 Exportar color difuso.
    11.5 Exportar color especular.
    11.6 Exportar nivel de opacidad.
    11.7 Exportar índice de la textura difusa.
    11.8 Exportar índice de la textura de opacidad.
    11.9 Exportar camino de la textura.

12. Crear directorio de texturas.
    12.1 Extraer camino del fichero.
    12.2 Mover texturas.

13. Crear un fichero stk.

3.3.2 Requisitos no funcionales.

Usabilidad: Cualquier usuario operador del 3dstudio max.

Diseño e implementación: Debe utilizarse una arquitectura por capas donde exista una capa superior en lenguaje c++ orientado a objetos y otra inferior en lenguaje c++ que utiliza la librería **IGAME** y **SDK** del 3dStudioMax.

Legales: Se regirá por las normas ISO 9000.

Soporte: Compatible con versiones de 3d Studio Max desde la 6 hasta la 8 que corren sobre sistemas operativos Microsoft Windows NT versión 4.0 en adelante.

Software: 3d Studio Max versión desde la 6 hasta la 8 que corre en los sistemas operativos, Windows XP Home Edition SP2, Windows 2000 SP4, Windows XP Profesional SP2 (recomendado), además es necesario tener instalado DirectX 9.0c.

Hardware: Tarjeta RAM 256 en adelante.

3.4 Modelo de casos de usos del sistema.

En esta sección se muestran los casos de usos de sistema divididos de acuerdo a etapas importantes en cuanto al proceso de exportación del fichero, se conciben los casos de uso del sistema así como los actores que van a interactuar con cada uno de ellos. Además, se seleccionan los casos de uso correspondientes al ciclo de desarrollo para hacerles sus especificaciones textuales en formato expandido.

3.4.1 Actores del sistema.

Los actores representan entidades que interactúan con el sistema, un actor del sistema es aquel que se beneficia con los resultados de las funcionalidades del mismo.

Actores	Justificación
Diseñador	Es el que se beneficia con las funcionalidades que brinda el sistema, en este caso exportar un fichero en formato stk.

Tabla 1: Descripción de actor del sistema

3.4.2 Casos de uso del sistema.

Los casos de uso son artefactos narrativos que describen, bajo la forma de acciones y reacciones, el comportamiento del sistema desde el punto de vista del usuario. Un caso de uso puede "incluir" la funcionalidad de otro caso de uso con su propio comportamiento, en el modelo de casos de uso de este sistema se pueden encontrar relaciones de este tipo.

A continuación se describen los casos de uso propuestos:

1. Exportar fichero.
2. Crear estructura de nodos.
3. Crear estructura geométrica.
4. Crear estructura de materiales.

CU-1	**Exportar fichero**
Actor	Diseñador
Descripción	Exportar toda la información referente a una escena.
Referencia	R5, R8, R11, R13, CU – 2, CU – 3, CU – 4.

Tabla 2: Descripción del CU Exportar fichero

CU-2	**Crear estructura de nodos**
Actor	CU “Exportar fichero”
Descripción	Crear la estructura de nodos genéricos para almacenarla en el fichero stk.
Referencia	R1, R6, R7.

Tabla 3: Descripción del CU Crear estructura de nodos

CU-3	**Crear estructura geométrica**
Actor	CU “Exportar fichero”
Descripción	Crear una estructura de geometrías libre de instancias, para almacenarla en el fichero stk.
Referencia	R2, R3, R4.

Tabla 4: Descripción del CU Crear estructura geométrica

CU-4	**Crear estructura de materiales**
Actor	CU “Exportar fichero”
Descripción	Crear una estructura de materiales, para almacenarla en el fichero stk.
Referencia	R9, R10, R12.

Tabla 5: Descripción del CU Crear estructura de materiales

3.4.3 Diagrama de casos de uso del sistema.

El diagrama de casos de uso del sistema describe la funcionalidad propuesta del nuevo sistema, basándose en la captura de los requisitos funcionales, y muestra las relaciones entre los casos de uso que representan las funcionalidades o principales procesos del sistema y los actores que interactúan con el mismo.

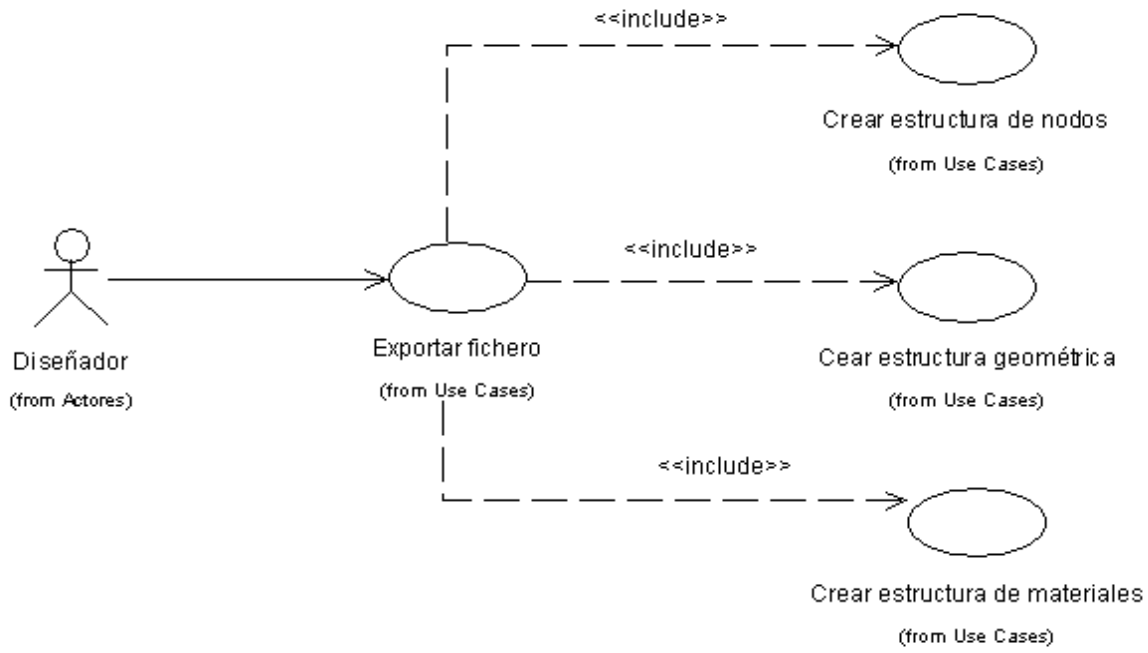

Fig 16: Diagrama de CU del Sistema

El Diseñador inicia el CU “Exportar fichero” que se encarga de exportar toda la información de una escena, y a su vez divide el caso de uso en tres sub-procesos, donde se encuentran el CU “Crear estructura de nodos” que es el encargado de almacenar la información de los nodos de la escena, el CU “Crear estructura geométrica” el cual almacena la información referente a las geometrías de la escena y por último el CU “Crear estructura de materiales” que es quien almacena la información de los materiales y las texturas en la escena.

3.4.4 Descripción de los casos de uso en formato expandido.

Para entender la funcionalidad asociada a cada caso de uso no es suficiente con la representación gráfica del diagrama de casos de uso, por esto se hace la expansión de mismo, que muestra más detalles de su funcionalidad y con esto se logra un mejor entendiemiento del mismo.

Caso de uso	
CU 1	**Exportar fichero.**
Propósito	Exportar toda la información referente a una escena.
Actores Diseñador	
Resumen: El CU se inicia cuando el Diseñador solicita exportar datos de una escena. Solicita la opción de exportar fichero .stk. Selecciona el camino donde se va a almacenar el fichero. Entonces se procede a almacenar toda la información que contiene la escena.	
Referencias	R5, R8, R11, R13, CU–2, CU–3, CU–4.

Acción del actor	**Respuesta del sistema**
1- Selecciona la opción exportar fichero .stk.	
	2- Muestra una ventana con las opciones de exportar objetos ocultos y seleccionar coordenadas del sistema que están en uso.
3- Selecciona las opciones que desee y selecciona la opción exportar.	
	4- Se verifica la existencia de nodos en la escena.
	5- Si no existen nodos en la escena ir al flujo alternativo “Escena vacía”.
	6- Se invoca al CU-2 “Crear estructura de nodos”.

	7- Se invoca al CU-3 “Crear estructura geométrica”.
	8- Se invoca al CU-4 “Crear estructura de materiales”.
	9- Se crea el fichero .stk.
	10- Se almacenan en el fichero las geometrías de la escena.
	11- Se almacenan en el fichero los nodos de la escena.
	12- Se almacenan en el fichero los materiales de la escena.
Flujo alternativo	
Escena vacía	
	5.1 - Se muestra el mensaje de error “Escena vacía”.
Postcondiciones	
Prioridad	Crítico

Tabla 6: Expansión del CU Exportar fichero

Caso de uso		
CU 3	**Crear estructura de nodos**.	
Propósito	Crear la estructura de nodos para almacenarla en el fichero stk.	
Actores CU “Exportar fichero”		
Resumen: El CU se inicia cuando es invocado por el CU “Exportar fichero”. Se recorren los nodos de la escena, se analiza la visibilidad del nodo, se hace un proceso de validación y se almacena en la lista de nodos.		
Referencias	R5, R6, CU-1	
Acción del actor		**Respuesta del sistema**
1- Invoca al CU-1 “Crear Estructura de Nodos”.		

	2- Se recorre los nodos de la escena.
	3- Si el usuario no marcó la opción de nodos ocultos ir al flujo alternativo “Nodos visibles”
	4- Se valida el tipo de nodo.
	5- Se almacena en una lista los nodos que contengan una geometría.
	6- Se identifica si es hijo o hermano del nodo anterior.
	7- Se almacena la información del nodo.
	8- Se crea un nodo genérico.
	9- Se almacena en la lista genérica de nodos.
Flujo alternativo	
“Nodos visibles”	
	3.1- Se analiza la visibilidad de los nodos.
	3.2 - Se seleccionan los nodos visibles.
	3.3 - Ir al paso 4.
Postcondiciones	
Prioridad	Crítico

Tabla 7: Expansión del CU Crear estructura de nodos

Caso de uso	
CU 2	**Crear estructura geométrica**.
Propósito	Crear una estructura de geometrías libre de instancias, para almacenarla en el fichero stk.
Actores CU “Exportar fichero”	
Resumen: El CU se inicia cuando es invocado por el CU “Exportar fichero”. Se comienza a recorrer la lista de nodos geométricos, por cada nodo se eliminan las instancias que contengan y se almacena en la lista de geometrías.	
Referencias	R1, R2, R3, R4, CU-1

Acción del actor	**Respuesta del sistema**
1- Invoca al CU “Crear Estructura Geométrica”.	
	2- Se recorre la lista de nodos geométricos.
	3- Se identifican las instancias de los nodos.
	4- Se almacena por cada instancia su ID y el ID del padre.
	5- Se eliminan las instancias.
	6- Se almacena la información de la geometría contenida en el nodo.
	7- Se crea una geometría genérica.
	8- Se almacena en la lista genérica de geometrías.
	9- Se ordena la lista genérica de geometrías (TriMesh y Polyline).
	10- Por cada nodo genérico se actualiza el índice de la geometría a la que apunta.
Postcondiciones	
Prioridad	Crítico

Tabla 8: Expansión del CU Crear estructura geométrica

Caso de uso	
CU 4	**Crear estructura de materiales**
Propósito	Crear una estructura de materiales, para almacenarla en el fichero stk.
Actores CU “Exportar fichero”	
Resumen: El CU se inicia cuando es invocado por el CU “Exportar fichero”. Se recorren los materiales que utilizan los nodos de la escena y se almacenan en una nueva lista de materiales y por cada material se valida la textura y se almacena en una lista.	
Referencias	R7, R8, R12, CU-1

Acción del actor	Respuesta del sistema
1- Invoca al CU “Crear Estructura de Materiales”.	
	2- Se recorren los materiales que usan los nodos de la escena.
	3- Se valida el material.
	4- Se almacena la información de un material.
	5- Se crea un material genérico.
	6- Se almacena en la lista genérica de materiales.
	7- Se recorre la lista de texturas por cada material.
	8- Se valida la textura.
	9- Se crea una textura genérica.
	10- Se almacena en la lista genérica de texturas.
	11- Se eliminan todas las instancias de las texturas.
	12- Se crea un directorio “ extura”.
	13- Se válida la existencia de una textura.
	14- Se copian las texturas al directorio “Textura”.
	15- Por cada material se actualiza el índice de la textura a la que apunta.
	16- Por cada nodo genérico se actualiza el índice del material al que apunta.
Postcondiciones	
Prioridad	Crítico

Tabla 9: Expansión del CU Crear estructura de materiales

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 y no funcionales de éste, y descritos los casos de uso que le permitirán al futuro usuario obtener los resultados esperados por el cliente.

4

Capítulo 4 : Diseño e implementación del sistema

Introducción

La primera parte del capítulo encierra el diseño del sistema propuesto. Se muestran los diagramas de clases agrupados por paquetes así como las relaciones entre los mismos, además se exponen los diagramas de secuencia de la realización de los casos de uso.

A continuación se pasa a la implementación del sistema. A partir del diseño de clases ya obtenido se crearán componentes físicos, que serán ficheros de extensión .h y .cpp debido a que se implementará haciendo uso del lenguaje C++. Además se elaborará el diagrama de componentes para el sistema.

4.1 Diagramas de clases de diseño.

Por la complejidad del Diagrama de clases de diseño y para su mejor entendimiento se han agrupado las clases en paquetes, el paquete **IGame_SDK** encapsula las clases que se utilizan de la librería IGame del 3dstudiomax y las clases que utiliza del SDK, encargadas de manejar los nodos de una escena, el paquete **Exporter** contiene la clase CSTKExporter y su relación con las clases del Igame_SDK. En el paquete **Node** es donde se encapsula las clases encargadas de almacenar la información referente a los nodos de la escena, el paquete **Geometry** encapsula las clases que se encargan del almacenamiento de la información de las geometrías y por último el paquete **Material** que contiene las clases encargadas de almacenar la información de los materiales y las texturas que se utilizan en la escena.

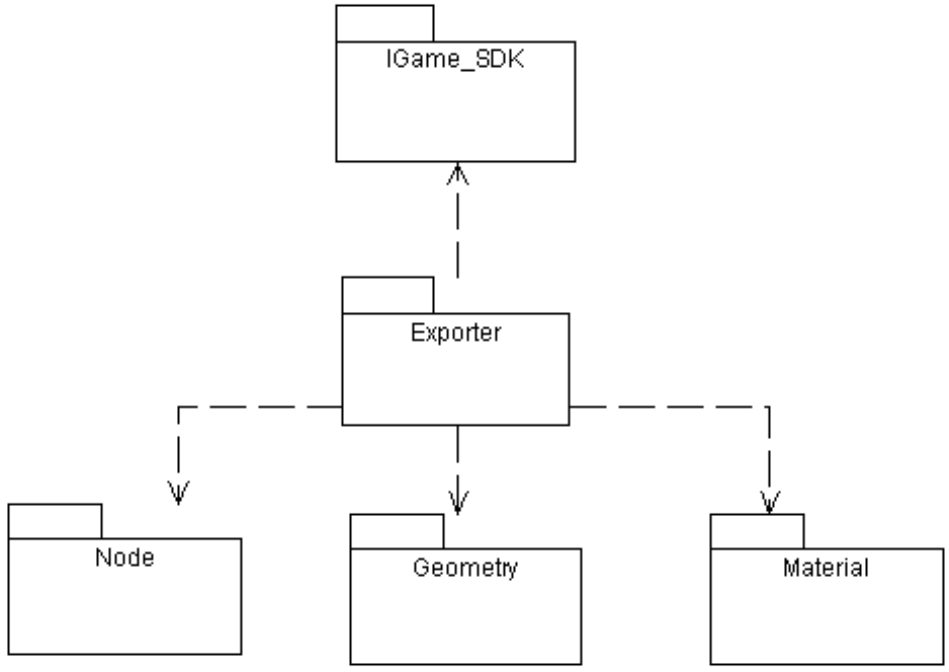

Fig 17: Diagrama de paquetes del diseño

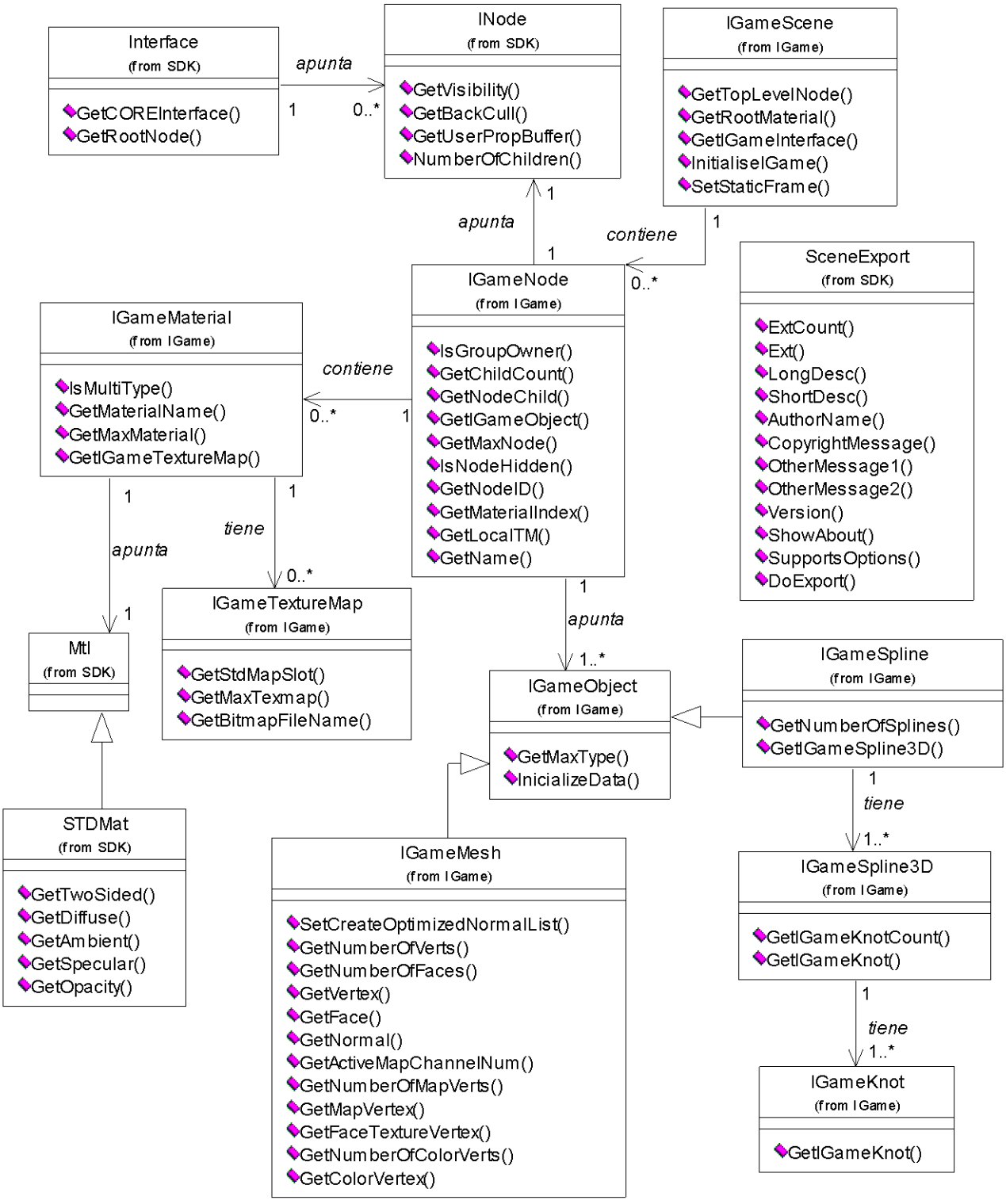

Fig 18: Diagrama de clases del paquete IGame_SDK

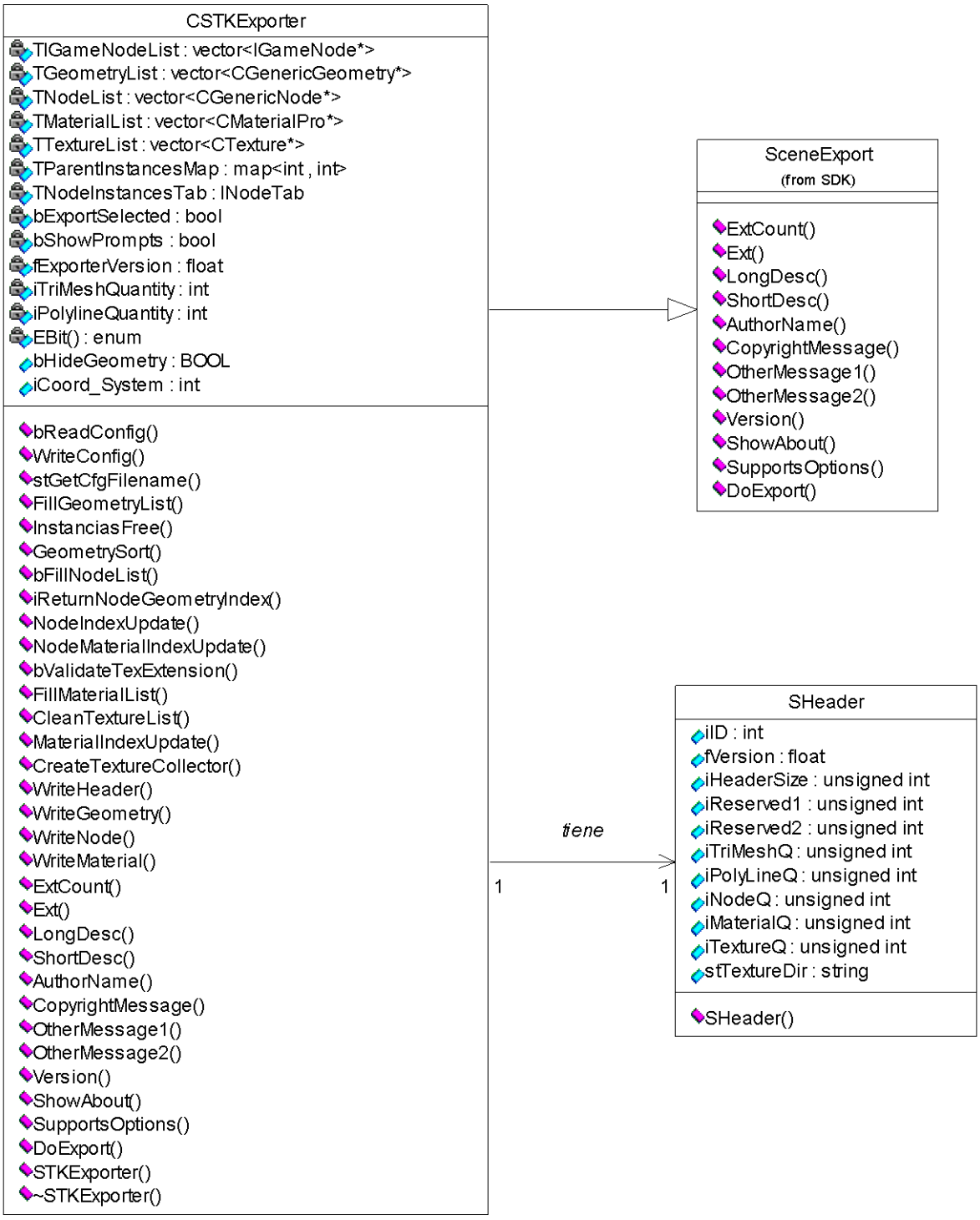

Fig 19: Diagrama de clases del paquete Geometry

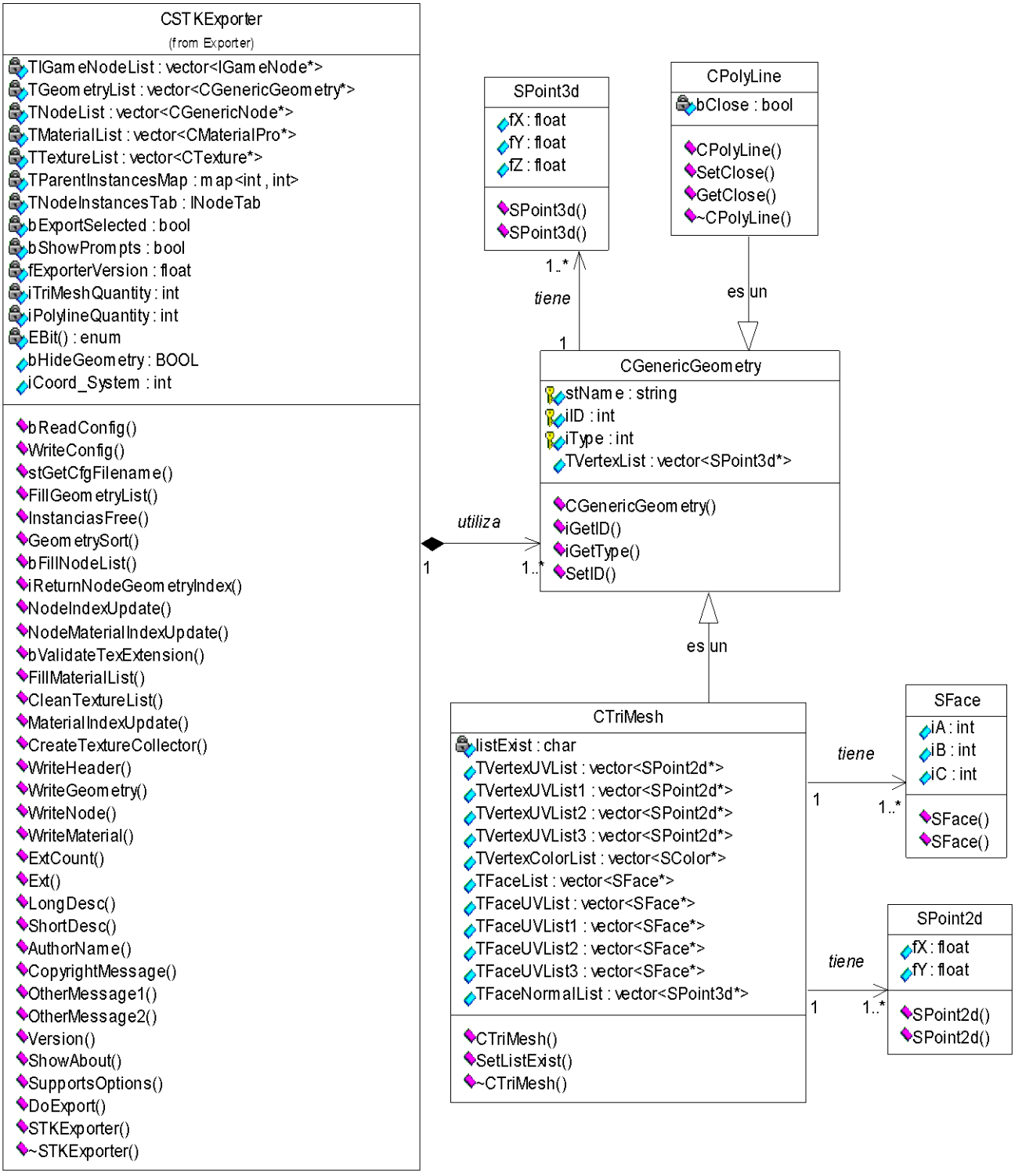

Fig 20: Diagrama de clases del paquete Geometry

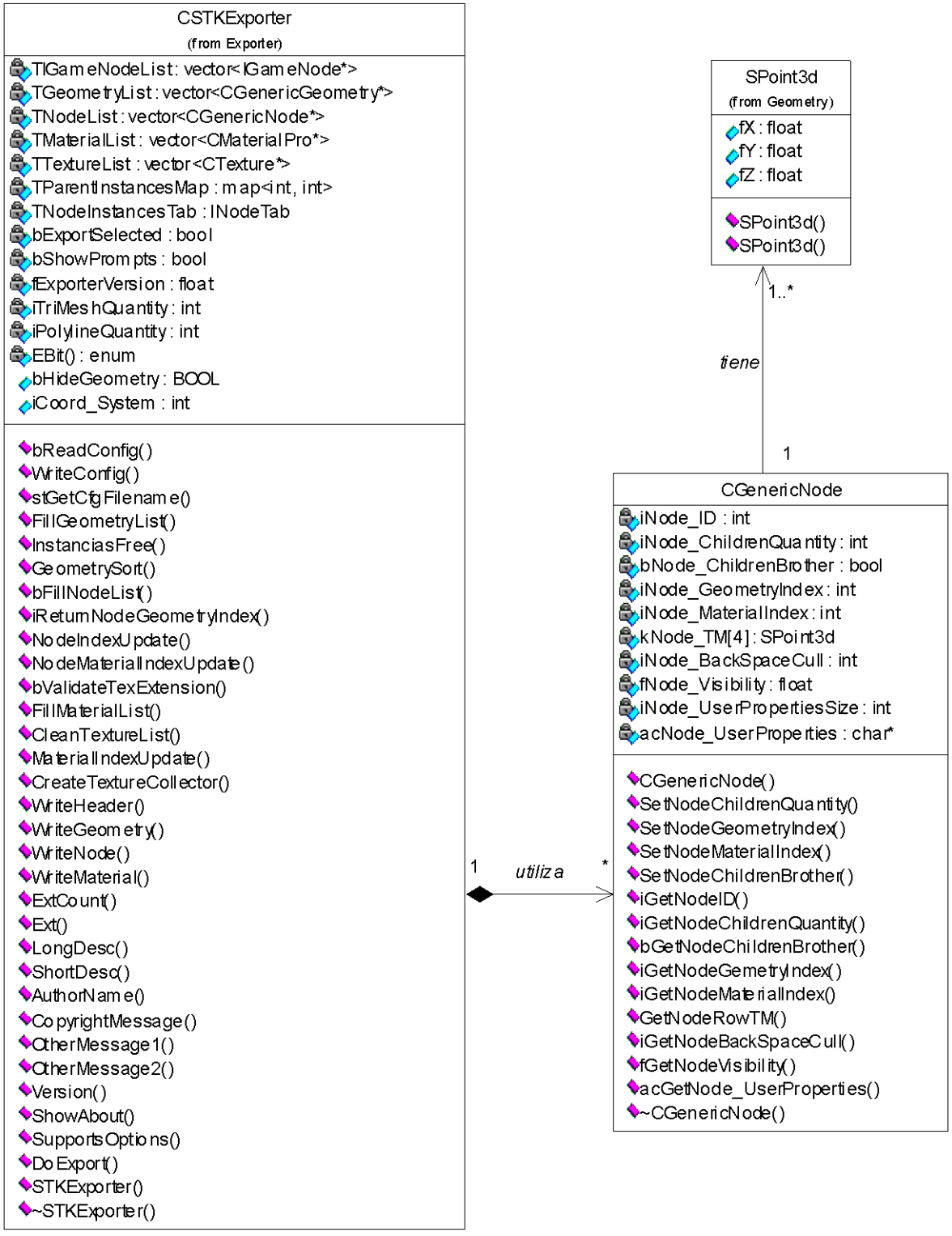

Fig 21: Diagrama de clases del paquete Node

Fig 22: Diagrama de clases del paquete Material

4.2 Descripción de las clases del diseño.

Las clases de diseño que se describen a continuación son las más importantes dentro del diseño de clases. Los métodos de acceso a miembros de clases (set y get) no son especificados en las descripciones, puesto que su funcionalidad es sencilla. Las clases del paquete IGame_SDK no se describen pues no son propias del fichero.

Nombre: CSTKExporter	
Tipo de clase: Controladora.	
Atributo	**Tipo**
TIGameNodeList	vector<IGameNode*>
TGeometryList	vector<CGenericGeometry*>
TNodeList	vector<CGenericNode*>
TMaterialList	vector<CMaterialPro*>
TTextureList	vector<CTexture*>
TParentInstancesMap	map<int , int>
TNodeInstancesTab	INodeTab
iCoord_System	int
bExportSelected	bool
bShowPrompts	bool
bHideGeometry	BOOL
fExporterVersion	float
iTriMeshQuantity	int
iPolylineQuantity	int
EBit {}	enum
Para cada responsabilidad:	
Nombre:	**bReadConfig()**
Descripción:	Lee la última configuración que quedó en la aplicación.
Nombre:	**WriteConfig()**
Descripción:	Escribe la configuración que dejó el usuario en el fichero de configuración.

Nombre:	**stGetCfgFilename()**
Descripción:	Carga el archivo de configuración.
Nombre:	**FillGeometryList(IGameNode*)**
Descripción:	Almacena la información de las geometrías de una escena en una lista de geometrías genéricas.
Nombre:	**InstanciasFree(IInstanceMgr*)**
Descripción:	Elimina todas las instancias de una geometría de la lista de geometrías.
Nombre:	**GeometrySort()**
Descripción:	Organiza la lista de geometrías dejando primero las tipo TriMesh y después los PolyLine.
Nombre:	**bFillNodeList(IGameNode*)**
Descripción:	Almacena toda la información de los nodos de una escena en una lista de nodos genéricos.
Nombre:	**iReturnNodeGeometryIndex(int)**
Descripción:	Retorna el índice de la geometría a la que apunta un nodo.
Nombre:	**NodeIndexUpdate()**
Descripción:	Actualiza el atributo índice de la geometría que tienen los nodos.
Nombre:	**NodeMaterialIndexUpdate()**
Descripción:	Actualiza el atributo índice del material que tienen los nodos.
Nombre:	**bValidateTexExtension(string&, string&)**
Descripción:	Comprueba que una textura es válida y almacena el camino en el primer parámetro de entrada y el nombre en el segundo.
Nombre:	**FillMaterialList(IGameMaterial*, int)**
Descripción:	Almacena toda la información de los materiales y las texturas que contengan una escena en una lista de materiales genéricos y texturas genéricas.
Nombre:	**CleanTextureList()**
Descripción:	Elimina todos los caminos de las texturas que contengan el mismo nombre.
Nombre:	**MaterialIndexUpdate()**
Descripción:	Actualiza el atributo índice de la textura que contiene cada material.
Nombre:	**CreateTextureCollector(string)**
Descripción:	Crea un directorio donde se almacena el fichero stk y hace una copia de

	las texturas que se usan en la escena al mismo.
Nombre:	**WriteHeader(FILE*, string)**
Descripción:	Escribe toda la información de la cabecera en el fichero stk.
Nombre:	**WriteGeometry(FILE*)**
Descripción:	Escribe toda la información de las geometrías en el fichero stk.
Nombre:	**WriteNode(FILE*)**
Descripción:	Escribe toda la información de los nodos en el fichero stk.
Nombre:	**WriteMaterial(FILE*)**
Descripción:	Escribe toda la información de los materiales y texturas en el fichero stk.
Nombre:	**Ext(int n)**
Descripción:	Extensión del fichero.
Nombre:	**ExtCount()**
Descripción:	Cantidad de extensiones que soporta.
Nombre:	**LongDesc()**
Descripción:	Descripción larga del plugins.
Nombre:	**ShortDesc()**
Descripción:	Descripción corta del plugins.
Nombre:	**AuthorName()**
Descripción:	Nombre del autor o autores.
Nombre:	**CopyrightMessage()**
Descripción:	Derechos de autor.
Nombre:	**OtherMessage1()**
Descripción:	Mensaje1.
Nombre:	**OtherMessage2()**
Descripción:	Mensaje2.
Nombre:	**Version()**
Descripción:	Versión del exportador.
Nombre:	**ShowAbout(HWND hWnd)**
Descripción:	Muestra una ventana con información de interés sobre la realización del plugins.
Nombre:	**SupportsOptions**
Descripción:	Muestra las extensiones que soportan el plugins.

Nombre:	**DoExport()**
Descripción:	Se encarga de procesar una escena y hace llamadas a métodos que permiten almacenar información de interés y exportarla en un fichero stk.
Nombre:	**CSTKExporter()**
Descripción:	Construye un objeto de tipo CSTKExporter.
Nombre:	**~CSTKExporter()**
Descripción:	Destruye y libera la memoria de un objeto tipo CSTKExporter.

Tabla 10: Descripción de la clase del diseño "CSTKExporter"

Nombre: CGenericGeometry		
Tipo de clase: Entidad		
Atributo		**Tipo**
stName		string
iID		int
iType		int
TVertexList		vector<SPoint3d*>
Para cada responsabilidad:		
Nombre:	**CGenericGeometry()**	
Descripción:	Construye un objeto de tipo CGenericGeometry.	

Tabla 11: Descripción de la clase del diseño "CGenericGeometry"

Nombre: CPolyLine	
Tipo de clase: Entidad	
Atributo	**Tipo**
bClose	bool
Para cada responsabilidad:	
Nombre:	**CPolyLine()**
Descripción:	Construye un objeto de tipo CPolylLine.
Nombre:	**~CPolyLine()**
Descripción:	Destruye y libera la memoria de un objeto tipo CPolylLine.

Tabla: Descripción de la clase del diseño "CPolyLine"

Nombre: CTriMesh	
Tipo de clase: Entidad	
Atributo	**Tipo**
TVertexUVList	vector<SPoint2d*>
TVertexUVList1	vector<SPoint2d*>
TVertexUVList2	vector<SPoint2d*>
TVertexUVList3	vector<SPoint2d*>
TVertexColorList	vector<SColor*>
TFaceList	vector<SFace*>
TFaceUVList	vector<SFace*>
TFaceUVList1	vector<SFace*>
TFaceUVList2	vector<SFace*>
TFaceUVList3	vector<SFace*>
TFaceNormalList	vector<SPoint3d*>
Para cada responsabilidad:	
Nombre:	**CTriMesh()**
Descripción:	Construye un objeto de tipo CTriMesh.
Nombre:	**~ CTriMesh ()**
Descripción:	Destruye y libera la memoria de un objeto tipo CTriMesh

Tabla 12: Descripción de la clase del diseño "CTriMesh"

Nombre: CTexture	
Tipo de clase: Entidad	
Atributo	**Tipo**
stT_Path	string
stT_Name	string
iT_IDType	int
Para cada responsabilidad:	
Nombre:	**CTexture()**
Descripción:	Construye un objeto de tipo CTexture.
Nombre:	**~CTexture()**
Descripción:	Destruye el objeto de tipo CTexture

Tabla 13: Descripción de la clase del diseño "CTexture"

Nombre: CMaterialPro	
Tipo de clase: Entidad	
Atributo	**Tipo**
iMp_ID	int
stMp_Name	string
stMp_TextureDiffuseName	string
stMp_TextureOpacityName	string
bMp_TwoSide	bool
kAmbientColor	SColor
kDiffuseColor	SColor
kSpecularColor	SColor
fMp_Opacity	float
iMp_IndexTexture	int
iMp_IndexOpacity	int
Para cada responsabilidad:	
Nombre:	**CMaterialPro()**
Descripción:	Construye un objeto de tipo CMaterialPro.
Nombre:	**~CMaterialPro(void)**
Descripción:	Destruye y libera la memoria de un objeto tipo CMaterialPro.

Tabla 14: Descripción de la clase del diseño "CMaterialPro"

Nombre: cNodeGenerico	
Tipo de clase: Entidad	
Atributo	**Tipo**
iNode_ID	int
iNode_ChildrenQuantity	int
bNode_ChildrenBrother	bool
iNode_GeometryIndex	int
iNode_MaterialIndex	int
kNode_TM[4]	SPoint3d
iNode_BackSpaceCull	int
fNode_Visibility	float
iNode_UserPropertiesSize	int
acNode_UserProperties	char*
Para cada responsabilidad:	
Nombre:	**CGenericNode()**
Descripción:	Construye un objeto de tipo CGenericNode.
Nombre:	**GetNodeRowTM(int il, SPoint3d& kV_point)**
Descripción:	Retorna una fila de la matriz de transformación y la almacena en una estructura SPoint3d.
Nombre:	**~ CGenericNode ()**
Descripción:	Destruye y libera la memoria de un objeto tipo CGenericNode.

Tabla 15: Descripción de la clase del diseño "CGenericNode"

4.3 Diagramas de secuencia.

A continuación se presentan los diagramas de secuencia correspondientes a la expansión de cada caso de uso planteados en el capítulo anterior, a partir de los mismos se puede tener una idea del tiempo de ejecución de cada algoritmo que interviene en los CU.

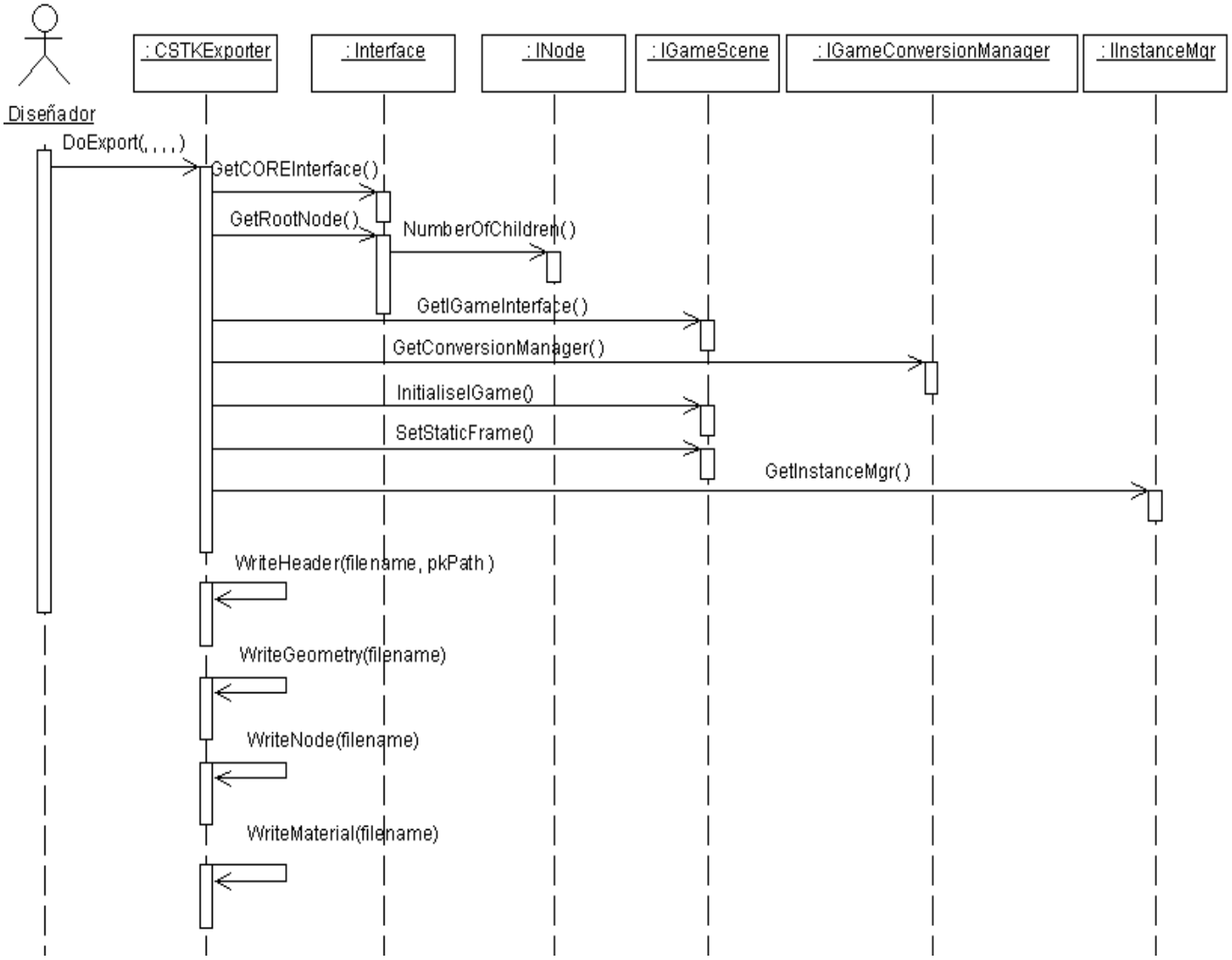

Fig 23: Diagrama de secuencia CU "Exportar fichero"

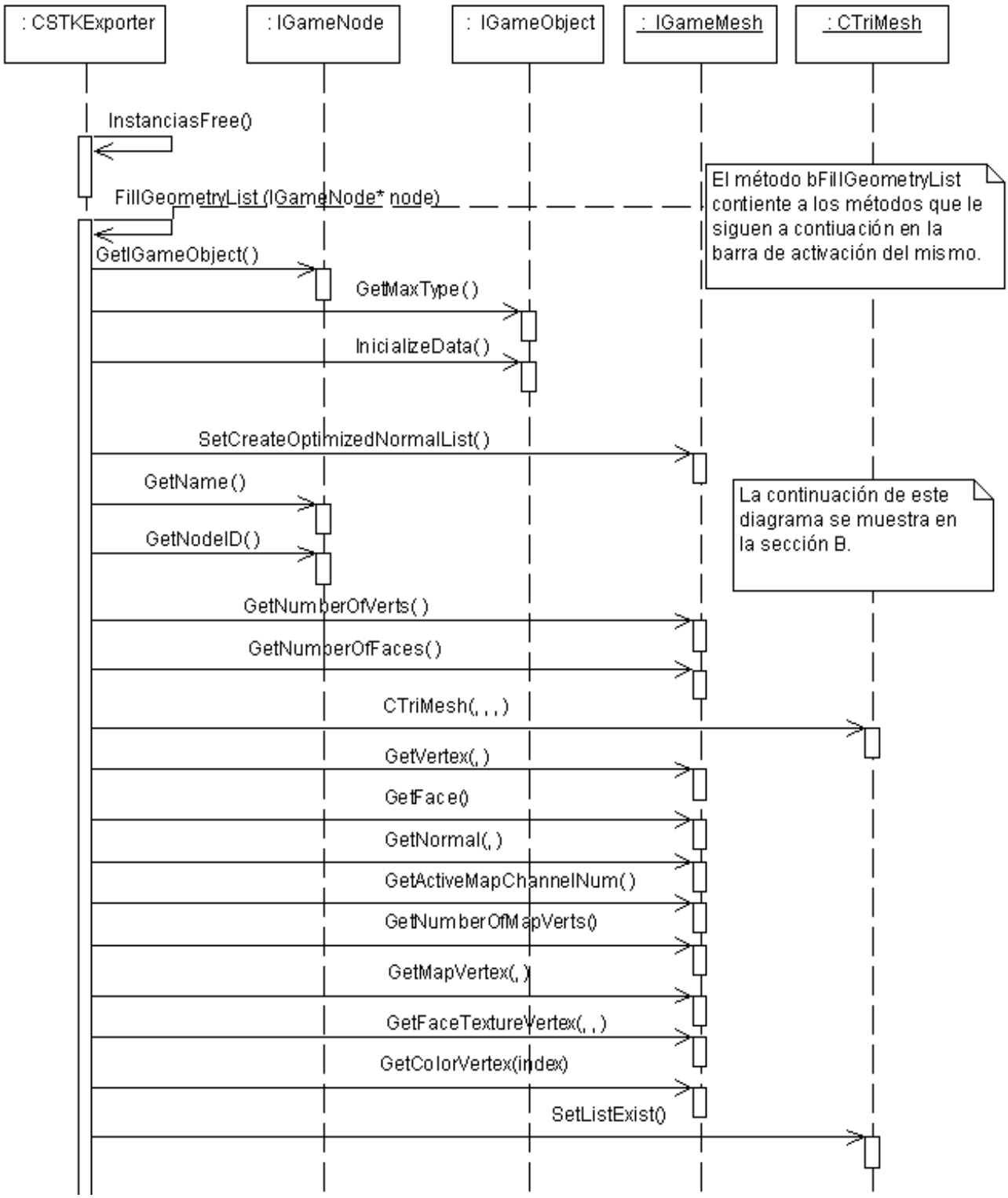

Fig 24: Diagrama de secuencia CU "Crear estructura geométrica" (A)

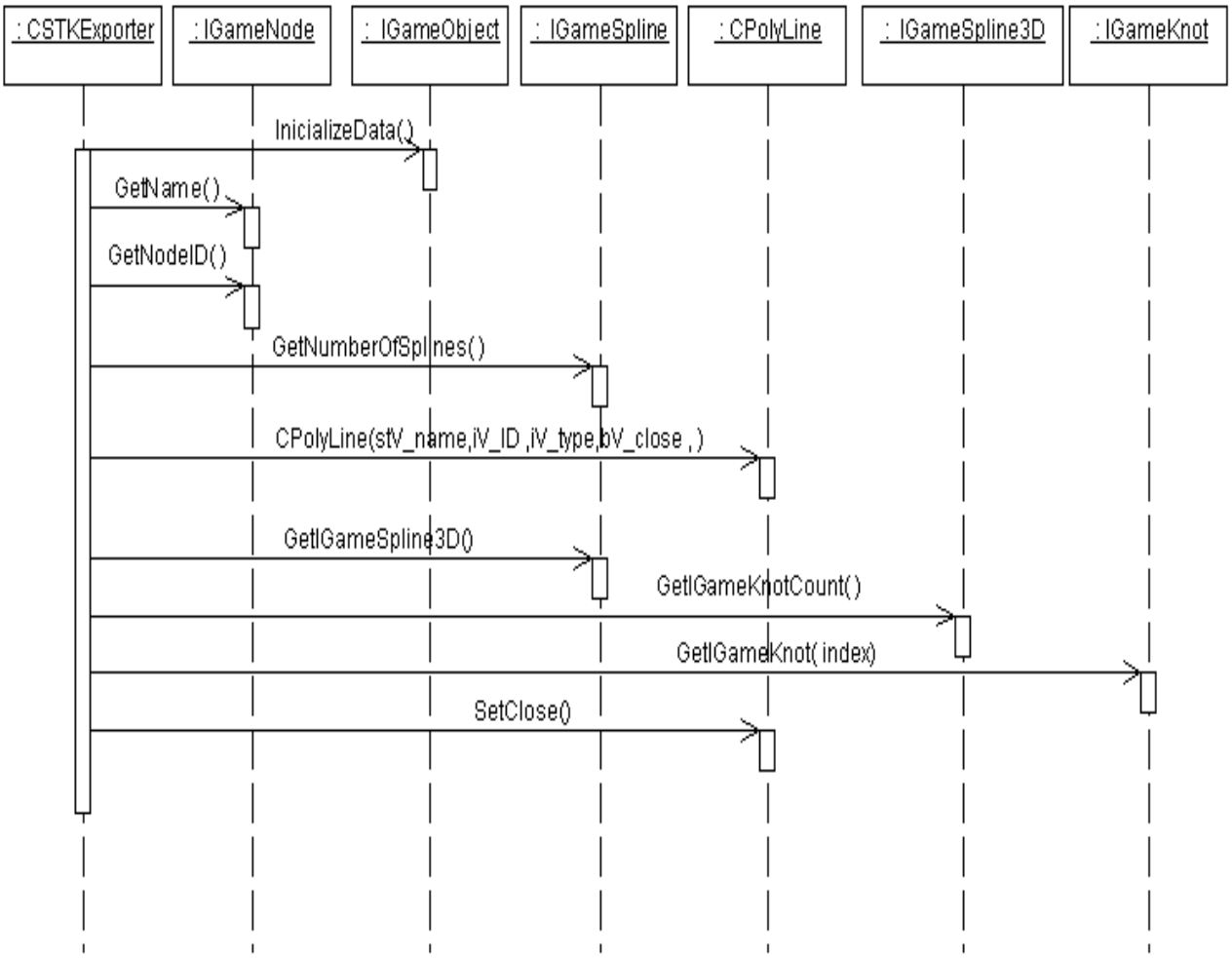

Fig 25: Diagrama de secuencia CU "Crear estructura geométrica" (B)

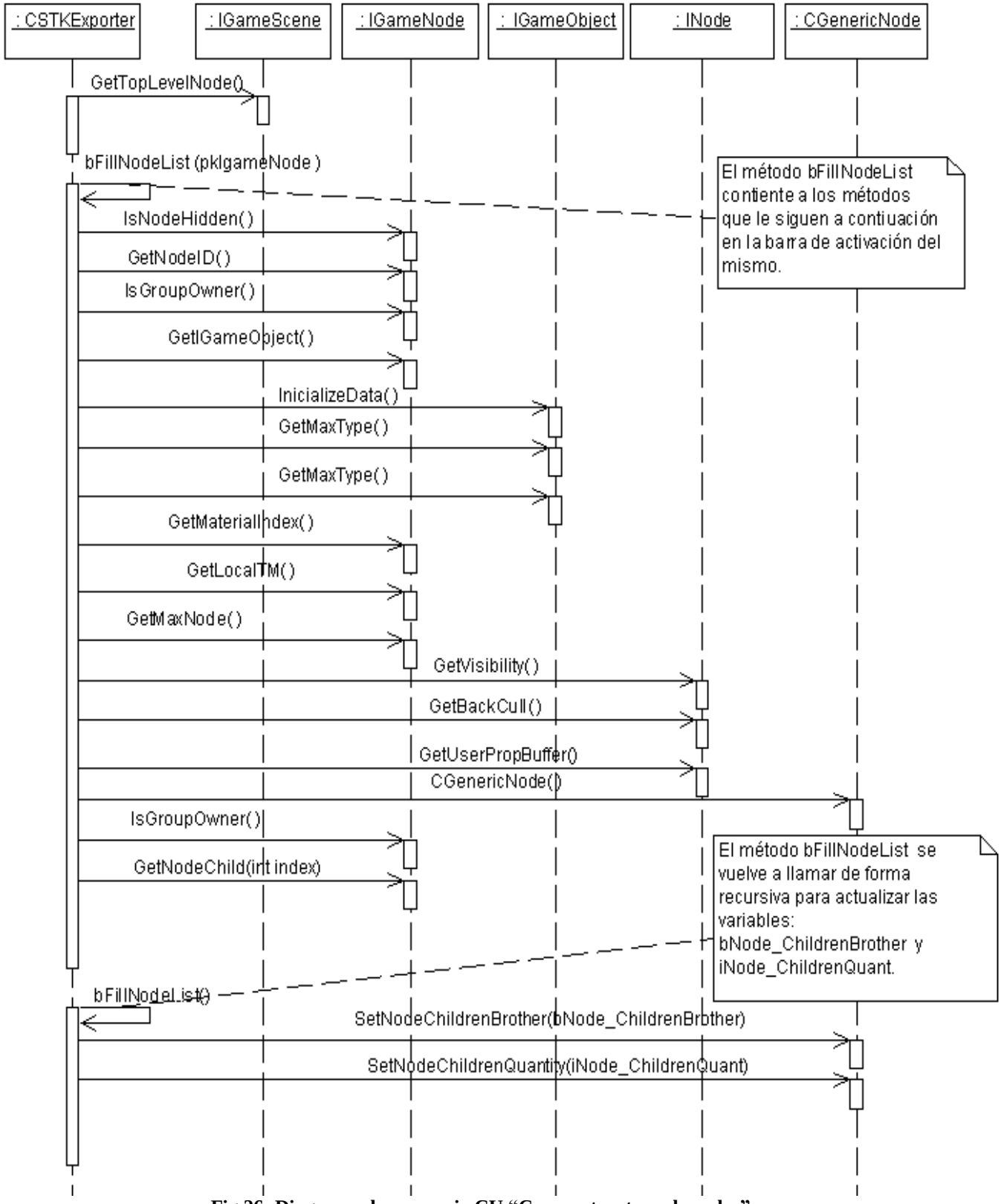

Fig 26: Diagrama de secuencia CU "Crear estructura de nodos"

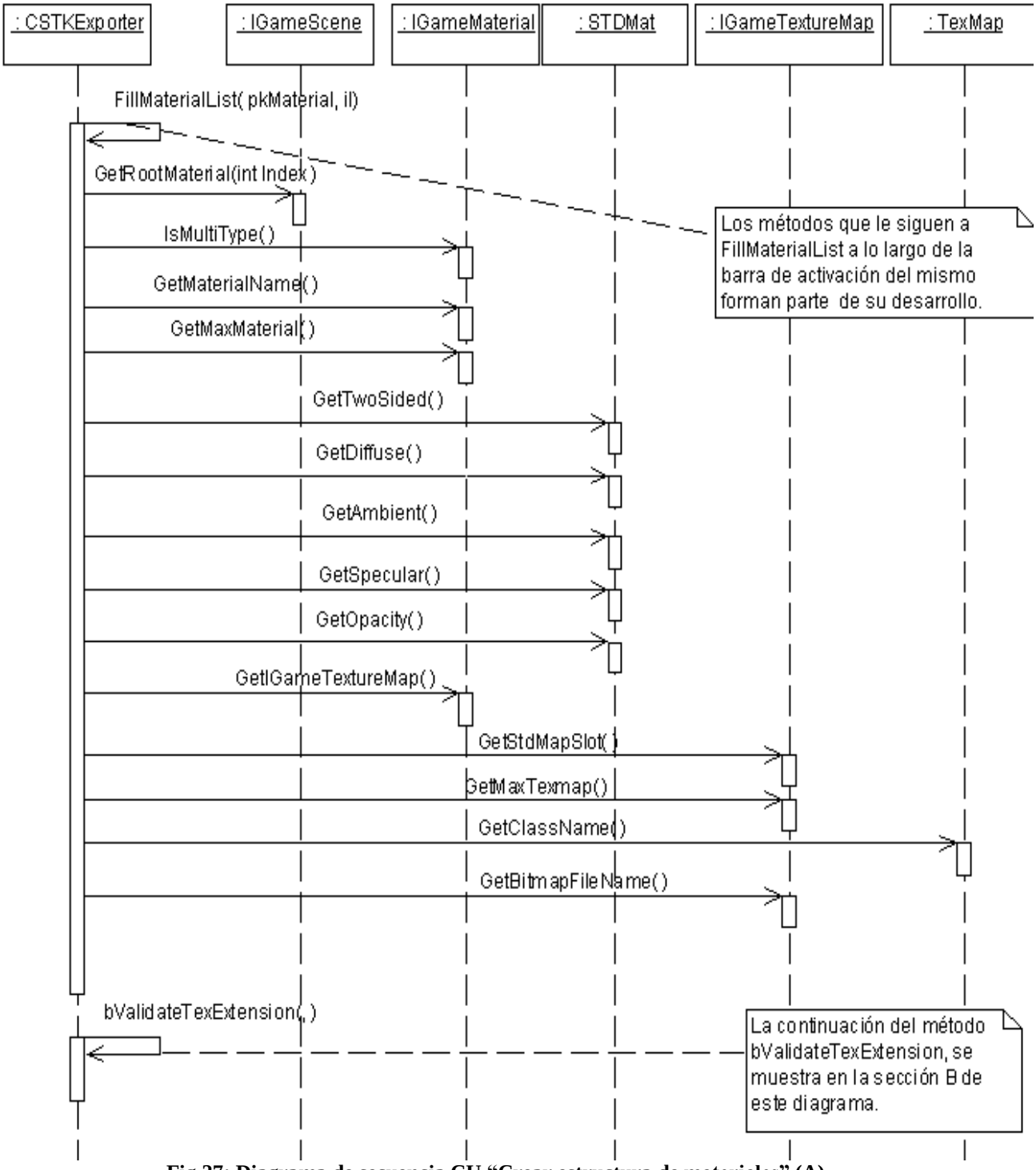

Fig 27: Diagrama de secuencia CU "Crear estructura de materiales" (A)

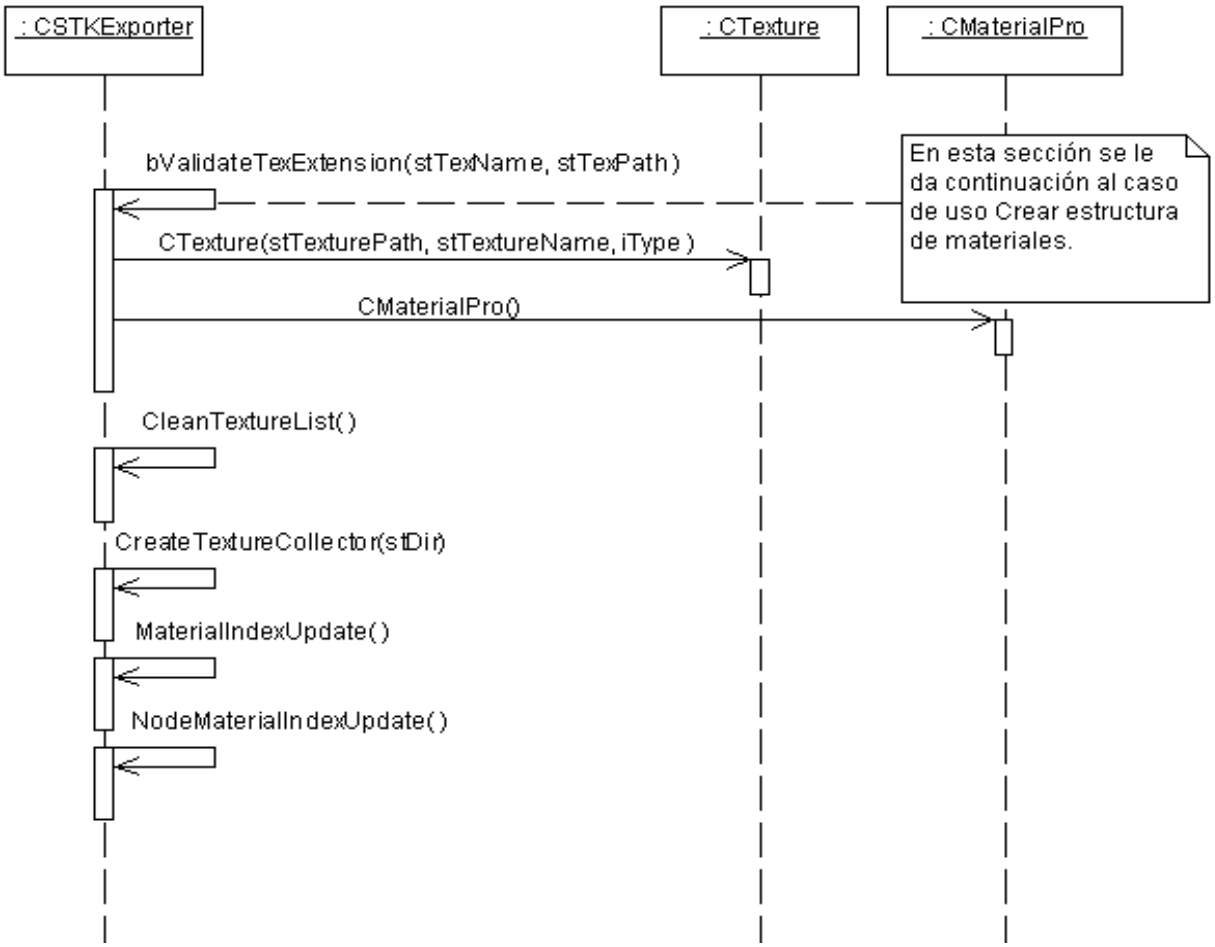

Fig 28: Diagrama de secuencia CU "Crear estructura de materiales" (B)

4.4 Estándares de codificación

El código del plugin sigue algunos estándares propuestos por el grupo de desarrollo (respetando los estándares de codificación para C++ (identado, uso de espacios y líneas en blanco, etc.)). Está programado en inglés, debido a que las palabras son simples, no se acentúan y es un idioma muy difundido en el mundo informático.

El conocimiento de los estándares seguidos para el desarrollo de la misma permitirá un mayor entendimiento del código, y es una exigencia de los autores que cualquier módulo que se añada debe estar codificado siguiendo estos estándares.

Nombre de los ficheros:

Los nombres de los ficheros .h y .cpp utilizan las iniciales del nombre de la herramienta (STK).

Ej: STKNameOfUnits.cpp

Tipos de datos:

Los tipos se nombrarán siguiendo el siguiente patrón:

Constantes: Las constantes se nombran con mayúsculas, utilizándose "_" para separar las palabras.

ej: const int MY_CONST_ZERO = 0;

Enumerados: Para los enumerados se utiliza el indicador "E" en el nombre del tipo, y en las constantes se utilizan las iniciales del nombre del enumerado. Las constantes van en mayúsculas.

```
Ej1: enum EMyEnum
{
ME_VALUE,
ME_OTHER_VALUE
};

Ej2: enum ENodeType
{
NT_GEOMETRYNODE,
NT_GROUPNODE...
};
```

Estructuras: Se utiliza el indicador "S" para indicar que es una estructura. Las variables miembros de la estructura se nombrarán igual que en las clases, leer más adelante.

ej: struct SMyStruct {…};

Clases:

Se utiliza el indicador “C” para indicar que es una clase. Ver más adelante la nomenclatura de las variables miembros.

ej: class CClassName;

Interfaces:

Se utiliza el indicador “I” para indicar que es una interfaz.

ej: IMyInterfaceName

Listas e iteradores de la std:

Para los tipos de datos utilizados de la librería estándar de C++ (vector, map, multimap, etc.), se utiliza el indicador “T”, con los sufijos List, Map y MultiMap según la estructura, así como el sufijo Iter para los iteradores. Además el nombre lleva el tipo de dato a almacenar en la estructura en cuestión:

Ej: vector<CNode> TNodeList;

TNodeList::iterator TNodeListIter;

Ej: map<> TNameMap;

TNameMap::iterator TnameMapIter;

Ej: multimap<> TNameMultiMap;

TNameMultiMap::iterator TNameMultiMapIter;

Declaración de variables:

Los nombres de las variables comienzan con un identificador 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 antepone el identificador “m_” (en minúscula), si son globales se les antepone el identificador “g_”, y en caso de ser argumentos de algún método, se les antepone el prefijo “arg_”.

Tipos simples:

```
Ej:     bool bVarName;
        int iName;
        unsigned int uiName;
        float fName;
        char cName;
        char* acName; // arreglo de caracteres
        char* pcName; // puntero a un char
        char** aacName; // bidimensional
        char** apcName; // arreglo de punteros
        bool m_bMemberVarName; // variable miembro de una estructura o clase
        char g_cGlobalVarName; // variable global
        short sName;
        void* pvName;
```

Instancias de tipos creados:

```
Ej:     EMyEnumerated eName;
        SMyStructure kName;
        CClassName kObjectName;  // objeto de una clase
        CClassName* pkName;      // puntero a objeto
        CClassName* akName;      // arreglo de objetos
        CClassName* m_akName; // variable miembro de clase
        IMyInterface* pIName;    // puntero interfaces
```

Métodos:

En el caso de los métodos, se les antepone el identificador del tipo de dato de devolución, y en caso de no tenerlo (void), no se les antepone nada. Los constructores y destructores, como lo exigen los compiladores, llevan el nombre de la clase.

Constructor y destructor:

Ej: CClassName (bool arg_bVarName, float& arg_fVarName);

~CClassName ();

Funciones:

Ej: bool bFunction1 (…);

int* piFunction2 (…);

CClassName* pkFunction3 (…);

Procedimientos:

Ej: void Procedure4 (…);

Métodos de acceso a miembros:

Los métodos de acceso a los miembros de las clases no se nombrarán “Gets” ni “Sets”, sino como los demás métodos, pero con el nombre de la variable a la que se accede y sin el prefijo “m_”:

Ej: Para la siguiente variable, los métodos de acceso serían:

```
intm_iMyVar; //variable

int iMyVar(); //”Get”

void MyVar(int arg_iMyVar) //”Set”

int& iMyVar(); //”Get” y ”Set”
```

4.5 Diagrama de despliegue.

El diagrama de despliegue para este sistema es muy simple, incluye sólo la representación de una PC, por esta razón quedó decidido que no es necesario representar dicho diagrama.

4.6 Diagrama de componentes.

Para lograr un mejor entendimiento del diagrama de componentes se dividió en dos vistas, una *vista binaria* donde se muestra la relación entre los componentes binarios y una *vista de código fuente* donde se encuentran los componentes .h y .cpp.

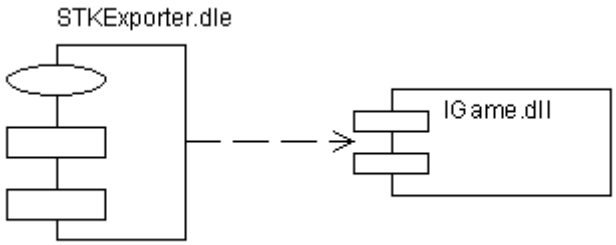

Fig 29: Diagrama de componentes de la *vista binaria.*

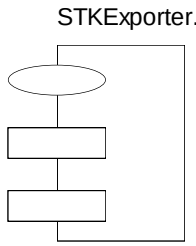

Fig 30: Componente binario STKExporter.dle

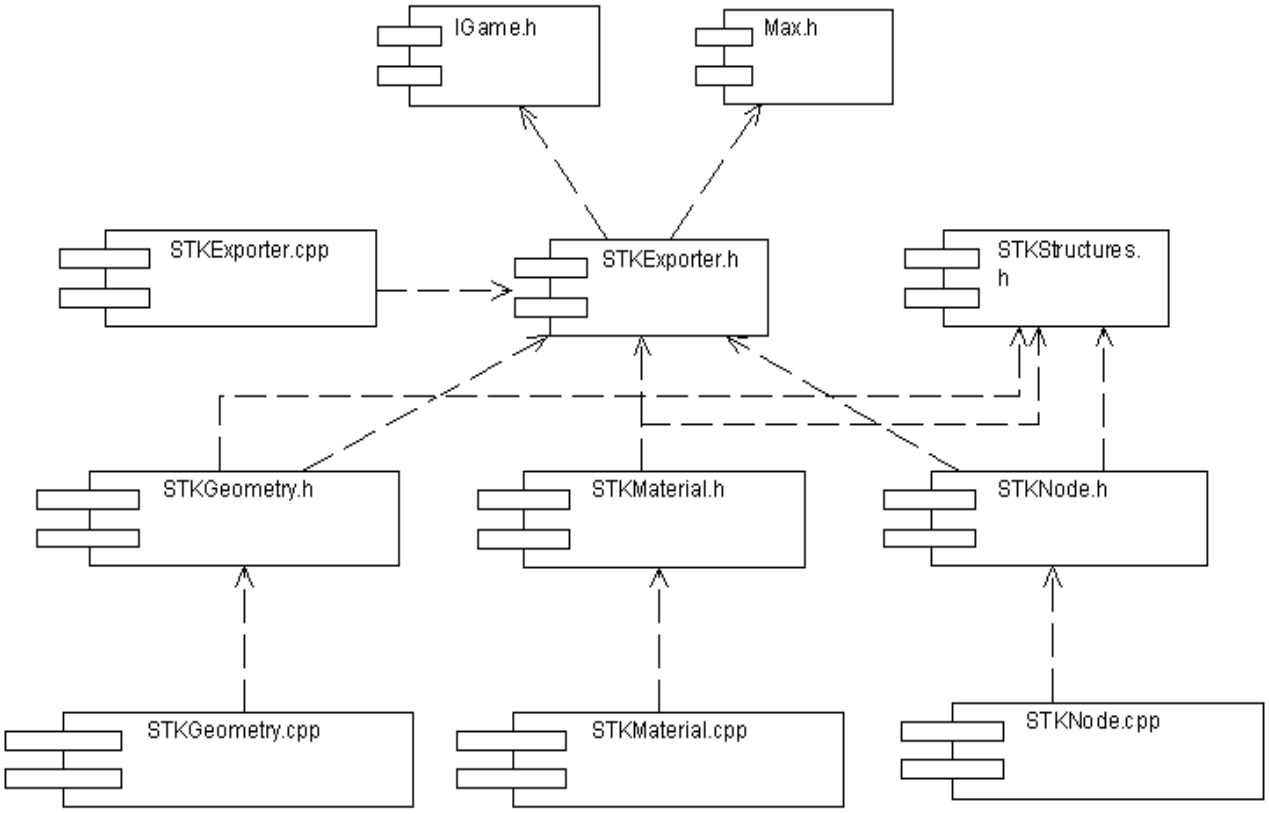

Fig 31: Diagrama de componentes de la *vista de código fuente*

Conclusiones

En este momento se encuentra todo preparado para pasar a la etapa de programación de los casos de uso. 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.

Conclusiones

Para el cumplimiento de los objetivos de este poyecto, en concordancia con las necesidades de la herramienta, fue necesario primeramente hacer un estudio de las técnicas, tecnologías y tendencias actuales en cuanto a la estructura de los formatos de ficheros, así como los diferentes algoritmos empleados para la generación de los mismos. En el estudio se analizaron las características de los formatos de ficheros gráficos 3d más usados en la actualidad y sus principales ventajas y desventajas. Además, a partir de esta investigación se proponen las características de las soluciones técnicas.

A continuación se realizó el proceso de Ingeniería del Software utilizando el Proceso Unificado del Software (RUP) como metodología de desarrollo, posteriormente se hizo la captura de los requisitos funcionales y no funcionales, y la identificación de casos de uso del sistema en conjunto con su descripción en formato expandido, se diseñaron las clases y se creó el diagrama de componentes que contendrá a las clases del sistema.

Finalmente, se obtuvo el formato de fichero que cumple con los requisitos de la herramienta y que permite al diseñador exportar entornos virtuales con una mejor organización en su estructura, lo que posibilita mayor velocidad a la hora de exportar y cargar el mismo. Además se acopló satisfactoriamente a la herramienta básica creada en el proyecto “Herramientas de Desarrollo para Sistemas de Realidad Virtual”.

Recomendaciones

Se recomiendan los siguientes aspectos al trabajo:

- Estudiar las diferentes técnicas y algoritmos para añadir las funcionalidades de exportar luces y cámaras como nuevos nodos.
- Estudiar las diferentes técnicas y algoritmos para añadir las funcionalidades de exportar animaciones por vértices y por huesos.
- Incluir un algoritmo de obtimización de vértices y coordenadas de vértices.
- Implementar el sistema para adaptarlo a nuevos software de diseño que funcionen en sistemas operativos libres como Linux.

Referencias bibliográficas

Libros

[1] Pipho, Evan. “Focus On 3D Models”
Premier Press. USA. 2003
[1.1]. Capítulo 3: Quake ll`s MD2 Models.
[1.2]. Capítulo 7: The 3ds Models.
[1.3]. Capítulo 8: MDL, Tehe Legendary Half-Life Format.
[1.4]. Capítulo 9: Enter the Quake: Quake lll`s MD3 Format.

[2] Wolfgang F. Engel. “Beginning Direct 3D Game Programming”
Premier Press.USA.2003
[2.1]. Capitulo 11: Working whith files.

Libros Digitales

[11] Autodesk [Consultado en: 2007] “*Weapons of Mass Creation*”
Disponible en: http://images.autodesk.com/adsk/files/games_brochure0.pdf

[12] Autodesk [Consultado en: 2007] “*Autodesk 3ds Max 8 SDK Help*”
Disponible en:
http://images.autodesk.com/adsk/files/3dsmax8sdkhelp_chm.zip

Documentos eléctronicos

[3] González Moreira, M. [Consultado en: 2006] Monografías.com
“Concepto de realidad virtual”. Disponible en:
http://www.monografias.com/trabajos11/realitual/realitual.shtml

[4] García Ruiz, M.A. [Consultado en: 2006]. "Conceptos de presencia e interactividad en objetos de aprendizaje de RV". Disponible en: http://docente.ucol.mx/~mgarcia/ponencias.html

[5] Wikipedia, La enciclopedia libre. [Consultado en: 2006]. "Formato de almacenamiento" Disponible en: http://es.wikipedia.org/wiki/Formato_de_almacenamiento

[6] Wikipedia, La enciclopedia libre. [Consultado en: 2006]. "STL (File Format)" Disponible en: http://en.wikipedia.org/wiki/STL_(file_format)

[7] Wikipedia, La enciclopedia libre. [Consultado en: 2006]. "Obj" Disponible en: http://en.wikipedia.org/wiki/Obj

[8] UnrealWiki, The Unreal Engine Documentation Site. [Consultado en: 2006]. "ASE File Format" Disponible en: http://wiki.beyondunreal.com/wiki/ASE_File_Format

[9] Galeano G., J.B. [Consultado en: 2006]. "La realidad virtual". Disponible en: http://www.monografias.com/trabajos4/realvirtual/realvirtual.shtml

[10] Activ@mente. [Consultado en: 2007] "VRML – Realidad Virtual". Disponible en: http://www.activamente.com.mx/vrml/

Tesis

[13] Camacho Román, Y. R.; Jiménez López, F. *Biblioteca Gráfica Para Sistemas de Realidad Virtual.* Tesis de pregrado inédita, Instituto Superior Politécnico"JOSÉ ANTONIO ECHEVERRÍA", 2004

Bibliografía consultada

1. Camacho Román, Y. R.; Jiménez López, F. *Biblioteca Gráfica Para Sistemas de Realidad Virtual.* Tesis de pregrado inédita, Instituto Superior Politécnico"JOSÉ ANTONIO ECHEVERRÍA", 2004
2. Autodesk, [Consultado en: 2006] "*Autodesk 3ds Max 8 SDK Help*" Disponible en: http://images.autodesk.com/adsk/files/3dsmax8sdkhelp_chm.zip
3. Pipho, Evan. "*Focus On 3D Models*" Premier Press. USA. 2003
4. Wolfgang F. Engel. "*Beginning Direct 3D Game Programming*" Premier Press.USA.2003

Glosario de abreviaturas

3D: Tres dimensiones.

3DS: 3D Studio (file format).

ASCII: American Standard Code for Information Interchange.

ASE: ASCII Scene Exporter

DIRECT X: Es una colección de APIs creadas para facilitar tareas relacionadas con la programación de juegos en la plataforma Microsoft Windows.

MD3: Nombre del formato que da Id Software al juego Quake 3.

OBJ: Nombre del formato creado por Alias Wavefront.

SDK: Software Development Kit*.*

SRV: Sistemas de Realidad Virtual.

STL: Standard Tessellation Language

SW*:* Software*.*

Glosario de términos

A:

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

Algoritmos: Conjunto finito de pasos o instrucciones que se deben seguir para realizar una determinada tarea.

*****

B:

Binario: En matemática el sistema binario es un sistema de numeración en el que los números se representan utilizando las cifras cero y uno ('0' y '1').

*****

C:

Canal difuso: Atributo por el cual se le pasa una textura o un color a un objeto.

Canales de materiales: Es la interfaz que brinda el 3D Studio Max para definir qué texturas tendrá un objeto y qué características presenta cada material del mismo.

*****

E:

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

*****

F:

Formas: Objeto compuesto por líneas y vértices en el espacio.

*****

I:

Instancias de un objeto: Es un grupo de objetos que mantienen las mismas características y tienen una estrecha relación, de tal manera que si ocurre un cambio en uno de ellos el resto sufre el mismo cambio.

*****

J:

Jerarquía de vértices: Es una lista ordenadas de vértices teniendo en cuanta algún tipo de prioridad.

*****

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 las ya existentes según una determinada transformación gráfica (rotación, trasladación, escalado y reflexión).

Modelo: Prototipo para la animación.

*****

N:

Normal: De manera básica es la dirección en la que mira cada cara de un objeto.

*****

P:

Poligonal: Del polígono o relativo a él.

Polígono: Figura geométrica plana limitada por segmentos rectos consecutivos no alineados, llamados lados.

Plugin: Fragmento de funcionalidades que se le agregan a un software.

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.

*****

S:

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

*****

T:

Textura: Imagen que sirve de “piel” a los modelos en un mundo virtual.

Texturizar: Aplicar las texturas.

*****

V:

Vector: Cantidad que expresa magnitud y dirección.

Vértice: Es un punto en el espacio dado por tres coordenadas x, y , z.