UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 5 ENTORNOS VIRTUALES

Herramienta para la creación de mapas a partir de un entorno 3D

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERÍA EN CIENCIAS INFORMÁTICAS

Autor: Yuremis Mengana Claro

Tutor: Ing. Yanoski Camacho Román

Ciudad de la Habana

2007

Dedicatoria

A mis padres. A Frank E.

A mi familia.

Yuremis

Agradecimientos

Agradezco a mi mamá por su apoyo incondicional, a Arian por siempre estar cuando lo necesité, a mi familia por toda su ayuda a lo largo de mi carrera, a mis compañeros y a todos aquellos que de alguna forma u otra contribuyeron al desarrollo de este trabajo.

Resumen

Los Sistemas de Realidad Virtual (SRV), se han expandido considerablemente a diferentes sectores de la sociedad en los últimos años. Esta tecnología puede encontrarse en aplicaciones de la defensa, la medicina, la educación y el entretenimiento, así como se han expandido los SRV también se han ido desarrollando y perfeccionando distintas herramientas para el trabajo con los gráficos 3D con el fin de minimizar tiempo y trabajo.

Este trabajo aborda el diseño de una herramienta que permite la creación de mapas para entornos tridimensionales. Para alcanzar este objetivo se estudian las principales técnicas de trabajo en la elaboración de herramientas gráficas.

La herramienta resultante de esta investigación permitirá a los diseñadores sustituir modelos existentes en el entorno por modelos que desde una vista superior represente al objeto al cual sustituye además permitirá eliminar cualquier modelo que entorpezca la visualización lo cual será de mucha utilidad a los diseñadores pues podrán trabajar con más rapidez y precisión, además que brindará al mapa un nivel de detalle mayor.

Palabras claves

Herramienta, mapa, entorno 3D.

Summary

Virtual Reality Systems (VRS) has been considerably expanded over different society sectors during the last years. This technology can be found in defense applications, medicine, educations and entertainment. As well as VRS has spread out, have been also developed and perfected different tools for working with 3D graphics to reduce time and work.

This work approach a design of a tool whish allows creating maps of 3D environments. To reach this goal, the main work techniques for creating graphics tools have been studied.

The final result of this investigation will allow to designers replacing existing models in an environment for models which represents the original object from a top view, also will be possible to eliminate any model that obstruct visualization. These features will be much useful to designers in order to work faster and more accurate, and will offer a more detailed map.

Contenido

Introducción

En el mundo los sistemas de realidad virtual se han extendido mucho por sus aplicaciones en distintas áreas, como medicina, ingeniería, arquitectura, educación, robótica, juegos electrónicos, televisión, etc.

En Cuba también se hace: ejemplo, la empresa SIMPRO que se dedica a la realización de simuladores profesionales como auto, tiro, entre otros, otro ejemplo es la UCI, que tiene entre sus perfiles la Realidad Virtual, y para esto, en colaboración con SIMPRO, tiene proyectos que se dedican a la investigación y producción de simuladores y juegos.

Uno de los simuladores producidos, el de conducción de auto, tiene un puesto para el instructor-evaluador en el cual, desde una PC, el instructor puede ver el desarrollo del ejercicio a través de un mapa acompañado de controles que muestran el estado de los mandos y parámetros del auto (velocidad, freno, etc.). En este mapa, se debe observar el entorno 3D, es decir, ver claramente las señales de tránsito (pare, semáforos, ceda el paso) desde una vista superior del entorno. Este mapa es hecho por los diseñadores, y cada vez que se hace un cambio en el entorno 3D, es necesario actualizar “a mano” el mapa 2D correspondiente, conllevando a más trabajo por parte de los diseñadores.

Se detecta la posibilidad de que otros simuladores futuros, así como los juegos, puedan contar con un mapa para orientar al evaluador o al propio usuario del sistema.

Por tanto, surge la **necesidad** de concebir una herramienta que automatice la creación de los mapas 2D a partir de entornos 3D, de forma que se cargue el entorno tridimensional, y se dé la posibilidad de tomar una vista superior y generar la vista 2D deseada, permitiéndose cambiar determinados elementos de dicho mapa de forma visual.

Teniendo como problema:

¿Cómo erradicar los problemas de diseño mediante una herramienta que sea interactiva con el usuario y automatice la creación de mapas?

Los procesos desarrollados para la creación de mapas son el **objeto de estudio** de este proyecto, y el **campo de acción**, la parte relacionada con la elaboración de mapas a partir de una herramienta de desarrollo para Sistemas de Realidad Virtual.

Este proyecto propone como **objetivo** general, desarrollar una herramienta que sea interactiva con el usuario, donde este pueda construir su mapa.

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

- Investigar las tendencias en cuanto al uso de mapas en sistemas de realidad virtual, como simuladores y juegos.
- Analizar los entornos de algunas herramientas de desarrollo visuales para diseño, como el 3D-Max.
- Estudiar las características de la herramienta básica existente en el proyecto para utilizarla como base de desarrollo de la herramienta a obtener.
- Analizar las potencialidades de las librerías gráficas OpenGL y Directx para capturar la información de lo que se esta visualizando.
- Desarrollar soluciones técnicas para alcanzar los objetivos propuestos.
- Analizar y diseñar una biblioteca de clases que dé solución a los problemas planteados.

Se espera obtener una herramienta que automatice la creación de los mapas que sea de fácil manejo por los diseñadores permitiéndole en poco tiempo obtener un mapa de mejor calidad que cumpla con las especificaciones requeridas.

El trabajo estará dividido en 5 capítulos, en el primero capítulo “Fundamentación Teórica” se realiza un estudio de los principales métodos y técnicas para el trabajo con los gráficos 3D, en el segundo “Soluciones técnicas” se muestran los elementos técnicos que fueron seleccionados para dar respuesta a la aplicación, en el tercero “Descripción de la Solución Propuesta” se analiza con mayor profundidad el objeto de estudio, se crea el modelo del dominio, se realiza la captura de requisitos, así como, una descripción de cada uno de los casos de uso que fueron definidos por los requisitos funcionales. En el cuarto “Análisis y diseño del Sistema” se presentan los diagramas de clases de diseño y los diagramas de interacción y en el quinto “Implementación del Sistema” se muestra el diagrama de componentes.

1

Capítulo 1 Fundamentación Teórica

Introducción

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

Existen determinadas características implícitas que debe cumplir el sistema, por citar algunos ejemplos, la posibilidad de aplicar texturas a objetos, la inserción y eliminación de modelos, la obtención de la imagen resultante entre otros.

En el presente capítulo se hace un análisis de los principales algoritmos, tendencias, técnicas, tecnologías, metodologías y *softwares* que se están empleando en el mundo para el trabajo con los gráficos 3D, así como los elementos principales con los que debemos trabajar para lograr las funcionalidades básicas que debe tener el sistema.

1.1 Técnicas, algoritmos y tendencias actuales

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

1.1.1 Representaciones de la superficie

En este trabajo se debe representar un entorno 3D es por ello que se debe conocer los tipos de representaciones que se utilizan en la vida real para representar las superficies.

Estas representaciones se hacen de varias formas: a través de mapas, planos, croquis, entre otros.

Mapas

Los mapas son representaciones planas de una parte de la superficie terrestre, que se los utiliza con mucha frecuencia. Existen varios tipos de mapas: físicos, políticos, climáticos, viales, etc. [13]

Fig. 1: Representación de mapa

Planos

Los planos son representaciones gráficas de una pequeña región, una ciudad o un barrio. En ella aparecen calles, avenidas, edificios, parques, etc. Los planos se utilizan también en la construcción. [13]

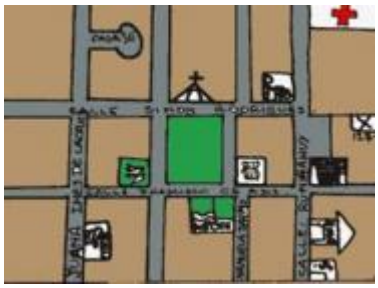

Fig. 2: Representación de plano

Croquis

El croquis es una representación gráfica muy elemental. Sirve para localizar lugares específicos, direcciones. Para su elaboración no se requiere instrumentos especializados ni medidas exactas. [13]

Fig. 3: Representación de croquis

1.1.2 Modelado geométrico

El término modelado geométrico se refiere a la colección de métodos usados para definir la forma y otras características geométricas del modelo que reproduce un objeto. Esto permite el trabajo con los objetos a partir de su dominio geométrico. [4]

Una escena puede contener distintos tipos de objetos (nubes, árboles, rocas, edificios, mobiliario, etc.) para los que existe una gran variedad de modelos de representación.

Los métodos de modelado geométrico son usados para construir con precisión matemática la descripción de la forma de un objeto real o para simular algunos procesos físicos y tecnológicos.

El modelado geométrico se usa para crear y comunicar información de la forma. Esto abarca la creación y mantenimiento del modelo del sólido para análisis y accesos futuros, por consiguiente el modelado del sólido constituye un aspecto importante en la simulación de fenómenos.

Se identifican tres aspectos distintos en la modelación geométrica: [4]

- *Representación*: referida a la forma física de un objeto. Se representa matemáticamente una aproximación del mismo.

- *Diseño*: es la creación y manipulación de formas a través de variables definidas.

- *Rendering*: proceso que transforma un modelo en una imagen real para su interpretación. En él intervienen tres aspectos, coloreado, iluminación y punto de vista.

Existen tres tipos de expresión del Modelado Geométrico que constituyen las formas principales utilizadas hasta hoy en la Gráfica por Computadoras: [4]

- Modelado de alambre (*wireframe*)

- Superficies de contorno (*boundary surface*)

- Modelos Sólidos (*solid*). [4]

Modelado de alambre

Los modelos de alambre representan los objetos a través de las aristas que delimitan sus superficies. Por consiguiente el modelo queda constituido por puntos, líneas y curvas. Presentan un aspecto poco realista, extremada simplicidad, trazado muy rápido, no requieren muestreo, ideales para tareas en las que no es imprescindible el realismo. [6]

La representación pura por alambre de un modelo sólido tridimensional posee tres inconvenientes principales que son: [4]

- Posibilidad de crear modelos ambiguos y objetos absurdos.
- Carencia de recursos gráficos o coherencia visual (línea de perfil o silueta que no son usualmente obtenidas desde el modelo).
- Posibilidad de que el modelo de alambre se aproxime al sólido en forma difusa.

Superficies de contorno

Las superficies de contorno están determinadas por una colección de curvas que unen puntos del contorno cuyas coordenadas son tomadas como continuas, con tres parámetros independientes en una función matemática X = x (t, u, v); Y = y (t, u, v); Z = z (t, u, v). [4]

Las superficies de contorno son usadas indistintamente a través de la historia. La superficie de contorno tiene un sentido más restringido, a través de las ecuaciones que se representan, no solamente se tienen todos los puntos de los elementos de contorno, sino también todos los puntos de su interior. [4]

Modelos Sólidos

Los sistemas de modelado de sólido deben brindar información para el análisis y la fabricación, o sea, deben tener en consideración características físicas y geométricas tales como: propiedades volumétricas del objeto, representación del objeto mediante proyecciones bidimensionales, conexión del objeto con otros elementos de una base de datos (jerárquico), generación de información para el control numérico. [4]

Con los modelos sólidos el hombre intenta reproducir cualquier objeto de la realidad, por tal motivo, existen varias formas de obtención que pueden cumplirse por separado o que para la obtención de un modelo pueden intervenir más de una. Las principales técnicas de obtención de modelos sólidos por separado son: [4]

- Geometría constructiva de sólidos (CSG siglas del inglés).
- Modelos de frontera.
- Modelos de barrido.
- Particionamiento espacial.

1.1.3 Transformaciones geométricas

Las transformaciones geométricas se utilizan para manipular objetos en el espacio 3D.

Lo primero que se debe hacer es utilizar un sistema de coordenadas que defina el espacio de forma numérica y proporcione una métrica que permita describir la distancia entre dos puntos. Se puede considerar la localización de los objetos con respecto a un punto central de referencia llamado origen. [7]

Las transformaciones más comunes son la traslación, escalado, rotación y reflexión. [12]

Traslación: Consiste en mover un objeto a una nueva posición. Las nuevas coordenadas vienen dadas por: $x' = x + T_X$, $y' = y + T_Y$, $z' = z + T_Z$.

Matriz:

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

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

Matriz:

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

Si $|S_X|>1$ y $|S_Y|>1$ y $|S_Z|>1$, aumenta el tamaño. Si $|S_X|<1$ y $|S_Y|<1$ y $|S_Z|<1$, disminuye. Si $S_X=S_Y=S_Z$, el escalado es uniforme, sino, no uniforme. Si $S_x<0$, el objeto se refleja respecto al plano Y_Z. Si $S_Y<0$, el objeto se refleja respecto al plano X_Z. Si $Sz<0$, el objeto se refleja respecto al plano X_Y.

Reflexión: Refleja el objeto respecto a un plano. Es una extensión del escalado, con alguno de los parámetros $S_X = -1$, $S_Y = -1$, $S_Z = -1$.

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

Matriz de rotación en X:

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

Matriz de rotación en Y:

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

Matriz de rotación en Z:

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

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

Matriz:

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

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

Concatenación de transformaciones

Se pueden combinar varias transformaciones para obtener operaciones más complejas, a través de la obtención de una matriz como producto de la multiplicación de las otras. Como las matrices son cuadradas se obtiene una única matriz. A estas matrices se les suele llamar SRT-Transform. (escalar-rotar-trasladar). [7]

El producto de matrices no es conmutativo ($M_1*M_2<>M_2*M_1$), por lo tanto, la aplicación de transformaciones tampoco lo es. [7]

Transformaciones que son conmutativas (el orden no importa): [7]

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

Transformaciones que no son conmutativas (el orden importa): [7]

- Traslación-Escalado.

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

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

Otro tipo de transformaciones es la llamada ***transformación no lineal***, que consiste en un conjunto de transformaciones que no se aplican de forma constante sobre todo el objeto. En lugar de utilizar constantes en las matrices de transformación se emplean funciones. A este tipo de transformaciones se les llama deformaciones globales. Entre las más conocidas se destacan: afilar (*Taper*), torcer (*Twist*) y curvar (*Bend*). [7]

1.1.4 Proyección

La proyección, se define como la representación de un cuerpo sobre un plano hecha según ciertas reglas geométricas. Los ejemplos de proyección más comunes son la sombra, y la proyección del ojo. [5]

Cada componente de la proyección se expresa de formas diferentes. El centro de proyección es un punto tridimensional (CP), la dirección de proyección es un vector espacial (DP), el plano de proyección se expresa a través de un punto y la normal (R_0, N) y los objetos según la estructura declarada por los programadores. [5]

Tipos de proyección

El primer nivel de clasificación de una proyección está dado por el lugar de origen de la proyección. Si el lugar de proyección se encuentra cerca del plano de proyección se puede decir que los rayos convergerán en un punto, dando origen a la proyección perspectiva. Si por el contrario el lugar de origen se encuentra en el infinito los rayos proyectantes pasarán paralelamente por los distintos puntos del objeto e incidirán en el plano de igual forma surgiendo la proyección paralela. [5]

La **proyección paralela** se clasifica en dos tipos: ortogonal y oblicua. Se dice que una proyección paralela es ortogonal cuando los rayos incidentes sobre el plano forman un ángulo de 90 grados con éste; en caso contrario es oblicua.

Este tipo de proyección es utilizada para la realización de los dibujos de planos técnicos de una casa, edificio, fábrica, equipo, pieza, etc. Debido a que el centro de proyección se encuentra en el infinito y los rayos proyectantes son paralelos entre sí, los cuerpos conservan su tamaño y forma originales. [5]

La **proyección perspectiva**: simula el comportamiento de nuestros ojos. En esta proyección no se conserva ni el paralelismo, ni el tamaño, ni la forma de los objetos originales. Mientras más alejados son los objetos, más pequeños son. Los objetos muy distantes pueden convertirse en un punto y los muy cercanos obstruirían la proyección hasta de ellos mismos por estar tan cerca. [5]

1.1.5 Texturizado

Texturizar es aplicar una textura (una imagen) a un modelo de manera que le sirva de “piel”, incrementando los detalles y el realismo de la escena significativamente. [3]

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

1.2 Potencialidades de librerías gráficas

En esta sección del capítulo se analizan dos de las bibliotecas gráficas que son muy populares para el desarrollo de gráficos 3D. [11]

Posteriormente se hace un análisis de las características de *OpenGL y DirectX*.

1.2.1 OpenGL

OpenGL es una librería gráfica que provee a los programadores de una interfaz de acceso al *hardware* (HW) gráfico. Consiste en alrededor de 120 comandos distintos usados para especificar los objetos y operaciones que se necesitan para producir aplicaciones tridimensionales interactivas. [10]

Es poderoso, con *rendering* a bajo nivel y una librería de *software* de modelamiento. OpenGL está diseñado para ser eficiente, disponible en diferentes plataformas y usado en cualquier aplicación gráfica. [1]

Arquitectura OpenGL

En el corazón del OpenGL está el ***rendering pipeline***, (tubería de *rendering:* serie de pasos que se siguen para aplicar el *render*, y que son manipulados por OpenGL), ver figura 4. [1]

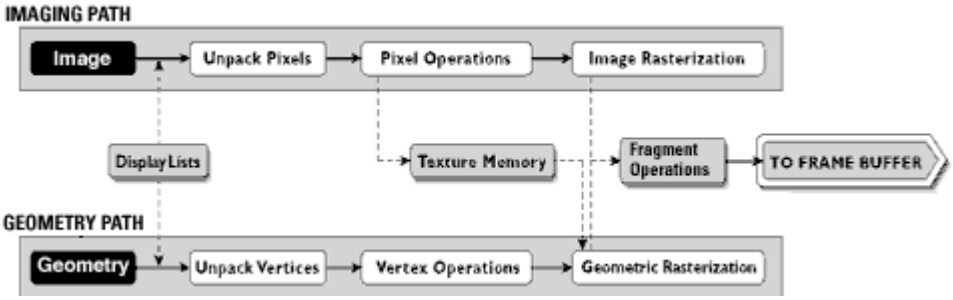

Fig. 4: Rendering Pipeline. Tubería de rendering.

Bajo Windows, OpenGL brinda una alternativa consistente en usar *Grafics Device Interface* (**GDI**, Interfaces de Dispositivos Gráficos), diseñada para hacer el HW gráfico completamente invisible al programador a través de capas de abstracción (esto aminora la velocidad). Las GDI se pueden ignorar y lidiar directamente con el HW gráfico (ver figura 5). [1]

Como OpenGL es una API gráfica, no soporta directamente ningún tipo de ventanas o menús, por lo que se necesitan manipular estos objetos. Cada sistema operativo usualmente tiene un grupo de extensiones que permiten este trabajo. En sistemas Unix, esto se logra con **GLX**. En Microsoft Windows, sin embargo,

se usa un grupo de funciones llamadas ***wiggle functions***. Cada una de estas funciones tiene prefijo **WGL** por *wiggle*. [2]

Windows define además un conjunto de funciones API Win32 que son usadas como interfaz con OpenGL. [2]

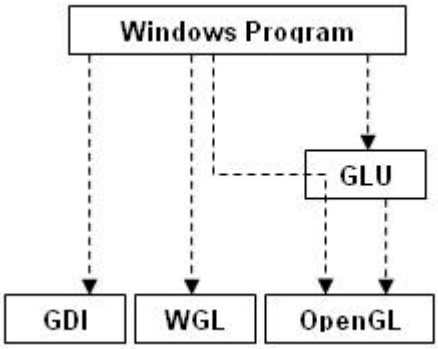

Fig. 5: Jerarquía OpenGL API bajo Windows.

OpenGL, ofrece además su propio grupo de librerías conocido como *OpenGL Utility ToolKit* (**GLUT**, Herramientas y Utilidades de OpenGL), con soportes disponibles en la mayoría de las plataformas y funcionalidades básicas en el área del trabajo con ventanas. GLUT mantiene la independencia de las plataformas, es decir, se puede mover fácilmente una aplicación basada en GLUT de Windows hacia Unix, con unos pocos cambios. [2]

OpenGL proporciona tres comandos básicos para manipular datos de imágenes:

- glReadPixels() – Lee un *array* rectangular de píxeles de un *framebuffer* y lo guarda en la memoria del procesador.
- lDrawPixels() – Escribe un *array* rectangular de píxeles de datos guardados en la memoria del procesador en el *framebuffer* en la posición actual del *raster*, especificada por glRasterPos*().

- glCopyPixels() – Copia un *array* rectangular de píxeles de una parte del *framebuffer* a otra. Este comando se comporta de manera similar a una llamada a glReadPixels() seguida de una llamada a glDrawPixels(), pero los datos nunca son escritos en la memoria del procesador.

1.2.2 DirectX

"DirectX es un intento de Microsoft de brindar un acceso "directo" al HW en el entorno del sistema operativo Windows, a través de un conjunto de APIs que controlan un grupo de funciones que acceden al HW o lo simulan si no existe." [1]

Dichas funciones incluyen soporte para aceleración gráfica 2D y 3D, control de dispositivos de entrada, funciones para mezclar y probar sonidos y música, control para juegos en red y *multiplayers*, y control sobre varios formatos de *streaming* de multimedia (*streaming* es un método de transferencia de datos continuamente, que permite mostrar los datos antes de que el fichero entero haya sido transmitido). [1]

Los componentes APIs que manipulan estas funciones son: DirectDaw, Direct3D, DirectInput, DirectSound, DirectMusic, DirectPlay, DirectShow. [1]

La filosofía de Microsoft es hacer una multimedia rápida, independiente de los dispositivos y rica en funciones para el sistema operativo Windows. [1]

Arquitectura DirectX

DirectX usa dos *drivers* o manejadores: la capa de abstracción de HW (HW *abstraction layer*, **HAL**) y la capa de simulador de HW (HW *emulation layer*, **HEL**; HEL ignora el HW e implementa sus funcionalidades). Cuando DirectX es inicializado, chequea el HW para ver si tiene ciertas potencialidades, y en dependencia de ello usa el HAL o el HEL. Esta arquitectura permite ser expandida fácilmente en un futuro HW (ver figura 8). [1]

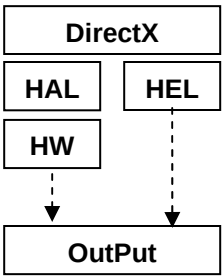

Fig. 1 Arquitectura HAL/HEL de DirectX.

DirectX ofrece funciones que permiten obtener el *surface* que no es mas que una matriz de píxeles que Direct3D utiliza normalmente para guardar datos 2D de imágenes. Esta superficie brinda funcionalidades como:

- **LockRect:** que nos permite obtener un apuntador a la memoria de la superficie. Luego podemos escribir y leer para cada píxel de la superficie. Además de esto previene que otras partes del programa accedan a ese pedazo de memoria mientras lo manejamos.

OpenGL contra DirectX

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

Ambas APIs usan la tradicional *grafics pipeline* (tubería gráfica). Es el mismo *pipeline* que se diseñó desde las primeras computadoras gráficas, que se ha ido modificando de acuerdo a los avances de HW aunque sin cambiar la idea básica. [1]

“Ambas APIs describen los vértices como un grupo de datos consistente en coordenadas en el espacio que definen la localización del vértice. Las primitivas gráficas (puntos, líneas, y triángulos) están definidos como un grupo ordenado de vértices. Sin embargo, la diferencia entre las APIs está en cómo los vértices son combinados para formar las primitivas: cada uno lo maneja diferente”. [1]

La siguiente tabla ofrece las características distintivas de ambas librerías ([1]).

Características	**OpenGL**	**DirectX**
Múltiples sistemas operativos	Sí	No
Mecanismo de extensión	Sí	Sí
Desarrollo	Múltiples Compañías	Microsoft
Textura de volumen	Sí	No
Z-Buffers independiente del *hardware*	Sí	No
Buffers de acumulación	Sí	No
Antialiasing de pantalla completa	Sí	Sí
Difuminación de movimiento	Sí	Sí
Profundidad de área	Sí	Sí
Rendering estéreo	Sí	No
Tamaño de punto y ancho de línea	Sí	No
Picking	Sí	No, pero con funciones de utilidad
Curvas y superficies paramétricas	Sí	No
Caché de geometrías	Listas de visualización	*Buffers* de vértices
Simulación de *Software*	Si el HW no está presente	Permite que la aplicación determine
Interfaz	Llamadas a	COM (*Component Object*

	procedimientos	*Model*)
Actualizaciones	Anuales	Anuales
Disponibilidad de código fuente	Implementación de muestra	Punto de inicio en Microsoft DDK (*Driver Development Kits*)

Tabla 1: Características distintivas OpenGL vs. DirectX.

Conclusiones

En el transcurso de este capítulo, como base para el entendimiento del tema en que se desenvolverá este proyecto, se mostraron las principales características de estos sistemas y las técnicas, tecnologías y tendencias actuales más utilizadas para su desarrollo.

2

Capítulo 2 Soluciones técnicas

Introducción

En el presente capítulo se proponen soluciones técnicas para el funcionamiento de la herramienta de creación de mapas a partir de una herramienta de desarrollo para Sistemas de Realidad Virtual.

2.1 Captura de imagen

Para esta primera versión solo se utilizará la biblioteca gráfica OpenGl por lo que la captura de la imagen se obtendrá a través de una función *glReadPixels()*, la cual copia la información de los píxels que hay en el buffer de cuadro a memoria. Junto a los argumentos de formato y tipo.

Sintaxis: *void glReadPixels(Glint x, Glint y, GLsizei ancho, GLsizei alto, GLenum formato, GLenum tipo, const GLvoid *pixels).*

Una ves que se obtenga la información de los píxel estos serán guardados en ficheros de extensiones (.bmp) o (.tga) según lo desee el usuario.

2.2 Consideraciones técnicas generales

La representación del entorno que se realizará será similar a los planos, pues se deben observar claramente las principales vías y algunas señales de transito.

El tipo de modelado será por alambre o mallas triangulares ya que son los más simples y la aplicación no requiere realismo. Estos modelos contendrán las coordenadas (u,v) para dar soporte a la aplicación de texturas a través de mapas de texturas.

Las transformaciones, se realizarán a través de matrices 3x3 y vectores, para las operaciones necesarias durante la actualización de los estados geométricos. No se realizarán transformaciones no lineales.

La aplicación se preparará para la librería gráfica, OpenGL, el resto del trabajo se hará a través de funciones programadas puramente en el lenguaje de desarrollo C++.

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

Conclusiones

Con este capítulo quedan sentadas las bases técnicas por las que se regirá el sistema. A partir de estos elementos se diseñará una estructura de clases en correspondencia con las técnicas citadas y para lograr los objetivos del proyecto.

3

Capítulo 3 Descripción de la Solución Propuesta

Introducción

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

3.1 Objeto de estudio

Es **objeto de estudio** de este trabajo son los procesos de creación de mapas a partir de entornos tridimensionales.

Específicamente la creación de mapas pero a partir de una herramienta de desarrollo para Sistemas de Realidad Virtual es el **campo de acción** de este trabajo, actualmente el mapa se realiza antes de elaborar el entorno con una herramienta para el trabajo con imágenes 2D como el *Photoshop* o el *Corel Draw* donde se ponen características de cada edificación, de manera que los desarrolladores del entorno pueden guiarse por la imagen hecha en esta herramienta y así elaborar el entorno, una ves terminado este, se ve de forma superior se le tira una foto y se comprueba cuan diferente es uno de otro y se arreglan los detalles, como se puede apreciar este proceso es poco preciso, no se puede obtener un nivel de detalle acorde y se pierde mucho tiempo en dicha elaboración, además que se hace casi imposible la visualización de las señales.

El **objetivo general** del presente trabajo es desarrollar una herramienta que sea interactiva con el usuario, donde este pueda construir su mapa dado un entorno.

3.2 Reglas del negocio

El fichero a cargar debe ser de extensión 3DX en caso de no ser se mostrará un mensaje indicando al usuario que cargue el entorno.

Las dimensiones de las imágenes que se utilizarán para texturizar las mayas, deben ser potencia de dos y estas imágenes se encontrarán dentro de la carpeta “Images” que se encontrará en la raíz de la aplicación.

Las imágenes a cargar deben tener los nombres siguientes: “Arbol.tga”, “Semaforo.tga”, “Techo.bmp”, “Ceda.tga”, “Pare.tga”.

Los elementos que serán sustituidos deben contener en su nombre los siguientes prefijos “edif” en caso de edificio, “pare” en caso de pare, “ceda” en caso de ceda el paso, “semaforo” en caso de semáforo, “techo” en caso de techo, “tree” en caso de árbol. Cualquier otro tipo de modelo que contenga en su nombre alguno de estos prefijos será tomado en cuenta como tal.

Los modelos que se deseen insertar solo será posible insertarlo si se encuentran sobre una superficie.

3.3 Modelo del dominio

Para una mejor comprensión del sistema, se desarrolló lo que se conoce como Modelo del Dominio.

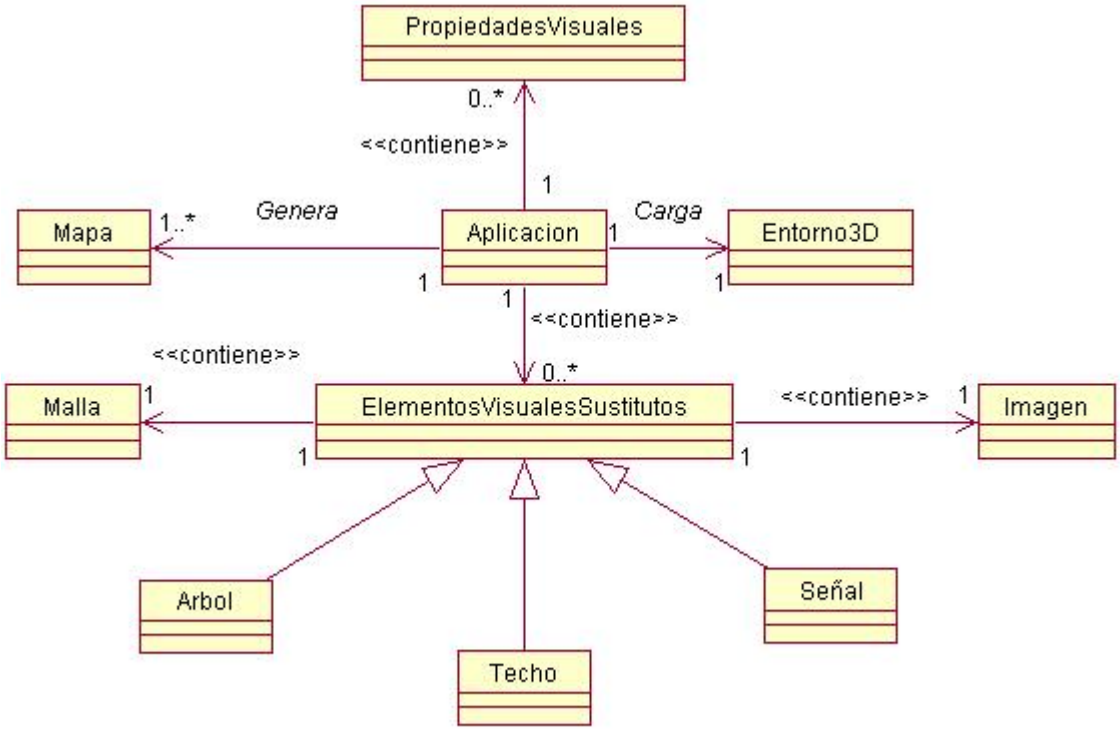

Fig. 6: Modelo del dominio.

En la Figura 1 se observa la relación de distintos conceptos que muestran la interacción del sistema. Se tiene una *Aplicación* que es la encargada de generar el *Mapa*, para esto es necesario que la *Aplicación* cargue el *Entorno3D* y luego transforme elementos del entorno cargado, por los *ElementosVisualesSustitutos* que no son mas que una *Malla* con la *Imagen* del objeto por el que se desea sustituir, estos objetos pueden ser *Árbol, Techo o Señal*, además la A*plicación* va a tener algunas *PropiedadesVisuales* que permitirá cambiar algunas configuraciones.

Glosario del modelo de dominio

Mapa: Fichero con la información del plano resultante que es generado por la aplicación.

Entorno3D: Fichero 3DX que será cargado por la aplicación.

PropiedadesVisuales: Algunas configuraciones que tiene la aplicación para facilitar el trabajo, como color de fondo.

ElementosVisualesSustitutos: Conjunto de modelos por los cuales se sustituirán los modelos reales estos pueden ser señales de tránsito, árboles o techos.

Malla: Modelo geométrico a partir de polígonos que tienen los ElementosVisualesSustitutos.

Imagen: Imagen que tendrán los ElementosVisulaesSustituos que representan las diferentes simbologías.

Árbol: Simbología que indica que en una posición dada hay un árbol.

Techo: Simbología que indica que en una posición dada hay un techo.

Señal: Simbología que indica que en una posición dada hay una señal de tránsito.

3.4 Captura de requisitos

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

3.4.1 Requisitos funcionales

1. Cargar entorno.
2. Eliminar entorno.
3. Crear árbol.
4. Crear techo.
5. Crear ceda paso.
6. Crear pare.
7. Crear semáforo.
8. Eliminar modelo.
9. Eliminar todos los árboles.
10. Eliminar todos los techos.
11. Eliminar todos los ceda el paso.
12. Eliminar todos los pares.
13. Eliminar todos los semáforo.
14. Eliminar todos los edificios.
15. Sustituir árbol.
16. Sustituir ceda el paso.
17. Sustituir pare.
18. Sustituir semáforo.
19. Sustituir techo.
20. Cambiar color de fondo.
21. Cambiar color de letras.
22. Mover arriba cámara.
23. Mover abajo cámara.
24. Mover izquierda cámara.
25. Mover derecha cámara.

26. Rotar cámara a la derecha.
27. Rotar cámara a la izquierda.
28. Acercar cámara.
29. Alejar cámara.
30. Salvar imagen en .bmp.
31. Salvar imagen en .tga.
32. Cargar imagen sustituta de pare.
33. Cargar imagen sustituta de árbol.
34. Cargar imagen sustituta de techo.
35. Cargar imagen sustituta de semáforo.
36. Cargar imagen sustituta de ceda el paso.
37. Inicializar posición de la cámara.
38. Insertar posición inicial.

3.4.2 Requisitos no funcionales

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

3.5 Modelo de casos de usos del sistema

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

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

Actor del sistema

Actores	Justificación
Diseñador	Es el que se beneficiará con las funcionalidades que brinda la herramienta, entre otras cosas podrá cargar desde ficheros, eliminar, insertar o sustituir los objetos y guardar imagen resultante.

Tabla 2: Actores

Casos de uso del sistema

Salvar imagen.

Cargar entorno.

Insertar modelo.

Sustituir modelo.

Eliminar modelo.

Inicializar aplicación.

Cambiar propiedades.

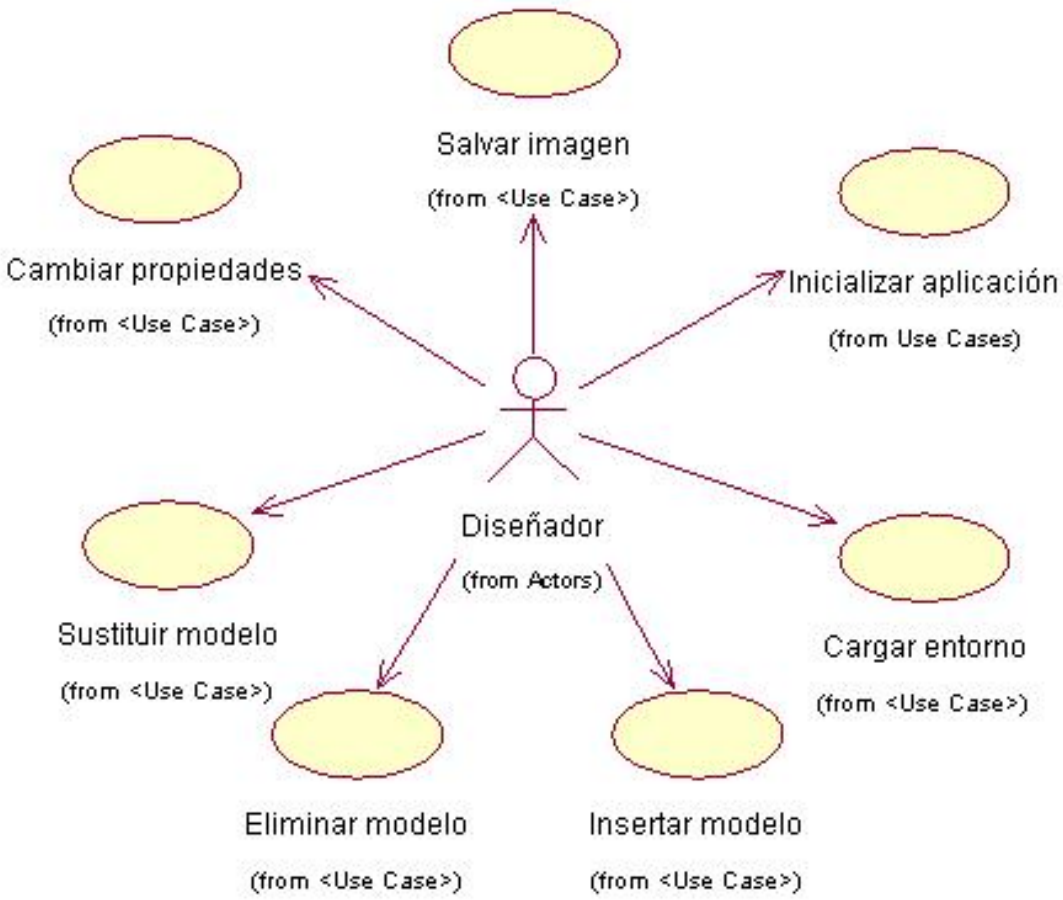

Fig. 7: Diagrama de CU sistema.

3.6 Especificación de los casos de uso en formato expandido

Nombre del caso de uso	**Cargar entorno.**
Actores	Diseñador
Propósito	Cargar el entorno tridimensional.
Resumen Se inicia cuando el Diseñador solicita del menú archivo la opción adicionar fichero, introduce los datos necesarios. El sistema realiza la acción y termina el CU.	
Referencias	RF 1 y 2
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1- El diseñador selecciona la opción archivo del menú	1.1 El sistema despliega el menú archivo y se muestra la opción "adicionar archivo".
2- El diseñador selecciona la opción "adicionar archivo".	2.1 El sistema muestra el formulario para adicionar el archivo.
3- El diseñador introduce los datos	3.1 Se elimina en caso de existir entorno anterior.
	3.2 Si la información es correcta se muestra el entorno que se seleccionó.
Curso alterno de los eventos	
Acción 3.2	Si los datos no son correctos el sistema muestra un mensaje indicando al usuario retornar a la acción 3
Precondiciones	
Poscondiciones	Queda el entorno cargado.
Prioridad	Crítico

Tabla 3: CU Cargar entorno.

Nombre del caso de uso	**Inicializar aplicación**
Actores	Diseñador
Propósito	Establecer valores iniciales.
Resumen Se inicia cuando el Diseñador solicita levanta la aplicación, el sistema se posiciona de forma superior y se cargan las imágenes prediseñadas y termina el CU.	
Referencias	RF 32 - 38
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1- El diseñador levanta la aplicación	1.1 El sistema posiciona la cámara hacia una vista superior
	1.2 El sistema guarda la posición que será tomada como posición inicial.
	1.3 El sistema carga las imágenes que representarán a los distintos modelos a sustituir.
Curso alterno de los eventos	
Acción 1.3	Si no se puede cargar alguna de las imágenes el sistema mostrará un mensaje indicando al usuario que las imágenes no se cargaron y no se podrán insertar o sustituir los distintos modelos.
Precondiciones	Deben existir las imágenes a cargar.
Poscondiciones	Se visualiza en el mundo desde una vista superior y quedan las imágenes prediseñadas cargadas.
Prioridad	Crítico

Tabla 4: CU Inicializar aplicación.

Nombre del caso de uso	**Salvar imagen.**
Actores	Diseñador
Propósito	Permite al diseñador guardar el resultado de las transformaciones en un fichero.
Resumen Se inicia cuando el Diseñador solicita del menú “Archivo” la opción (Salvar mapa como…). El sistema despliega una ventana para que el usuario inserte el nombre y tipo, se guarda la imagen y finaliza el CU.	
Referencias	RF 30 y 31
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1- El diseñador selecciona la opción “Archivo” del menú.	1.1 El sistema despliega el menú.
2- El diseñador selecciona la opción “Salvar mapa como…”	2.1 El sistema muestra un formulario para que el usuario inserte el nombre, extensión (.bmp, .tga) y la dirección.
3- El diseñador introduce los datos.	3.1 El sistema guarda la información del buffer de visualización en un fichero con la extensión deseada.
Curso alterno de los eventos	
Acción 2.1	Si la información no es correcta se muestra un error especificando el tipo de error e indica al usuario retornar a la acción 2.
Precondiciones	
Poscondiciones	Es capturada la imagen del mundo y guardada en un fichero de extensión .bmp o .tga.
Prioridad	Primario

Tabla 5: CU Salvar imagen

Nombre del caso de uso	**Sustituir modelo.**
Actores	Diseñador
Propósito	Permite al diseñador sustituir todos los modelos reales por modelos sustitutos.
Resumen Se inicia cuando el Diseñador solicita del menú Edición la opción sustituir. El sistema busca todos los modelos del tipo seleccionado guarda su posición los elimina y crea un modelo sustituto por cada uno eliminado, los traslada a cada una de las posiciones guardadas y finaliza el CU.	
Referencias	RF15 - 19
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1- El diseñador selecciona el menú "Edición".	1.1 El sistema despliega el menú.
2- El diseñador selecciona la opción "Sustituir".	2.1 El sistema muestra los tipos de modelo que se pueden sustituir.
3- El diseñador selecciona el tipo de modelo (árbol, señal, techo).	3.1 El sistema busca los modelos del tipo seleccionado y se guardan las posiciones (x,y,z) de cada uno.
	3.2 El sistema elimina los modelos encontrados.
	3.3 Se crea una malla por cada modelo eliminado.
	3.4 Se le asignan las posiciones(x,y,z) según se fueron guardando.
	3.5 El sistema busca la imagen a aplicar como textura.
	3.6 El sistema texturiza la malla con la imagen.
	3.7 Se visualizan.
Curso alterno de los eventos	
Acción 2.1	Si no existe se muestra un mensaje al usuario.
Precondiciones	Debe existir un entorno y las imágenes prediseñadas deben haber sido cargadas.
Poscondiciones	Quedan sustituidos los modelos que se seleccionaron a sustituir.
Prioridad	Secundario

Tabla 6: CU Sustituir modelo.

Nombre del caso de uso	**Eliminar modelo.**
Actores	Diseñador
Propósito	Permite al diseñador eliminar todos los modelos de un tipo (árbol, señal, techo) o eliminar solo uno.
Resumen Se inicia cuando el Diseñador solicita del menú Edición la opción “eliminar todos” o “eliminar”, se introducen los datos se busca el modelos o los modelos a eliminar. El sistema realiza la acción y finaliza el CU.	
Referencias	RF 8 - 14
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1- El diseñador selecciona la opción “Edición” del menú.	1.1 El sistema despliega el menú Edición.
Escenario 1: Eliminar todos	
1- El diseñador selecciona la opción “Eliminar todos”.	1.1 El sistema muestra los tipos de modelo que se pueden eliminar.
	1.2 Se buscan todos los modelos del tipo seleccionado.
	1.3 Se eliminan todos los modelos de ese tipo encontrados.
Curso alterno de los eventos	
Acción 1.2	Si no existe ninguno se le informa al usuario.
Escenario 2: Eliminar	
1- El diseñador selecciona la opción “Eliminar”.	
2- El diseñador selecciona el modelo dando un click encima del modelo.	2.1 El sistema verifica que se halla seleccionado un modelo.
	2.2 Se obtiene el id del modelo seleccionado.
	2.3 El sistema elimina el modelo seleccionado.

Curso alterno de los eventos	
Acción 2.1	Si no se seleccionó ningún modelo se muestra un mensaje que indica al usuario retornar a la acción 2.
Precondiciones	Debe existir un entorno.
Poscondiciones	Es eliminado el o los modelos seleccionados.
Prioridad	Secundario

Tabla 7: CU Eliminar modelo.

Nombre del caso de uso	**Insertar modelo.**
Actores	Diseñador
Propósito	Permite al diseñador insertar un modelo tipo (árbol, señal o techo).
Resumen Se inicia cuando el Diseñador solicita del menú “Inserción” la opción (árbol, señal, techo). El usuario selecciona la posición del modelo que desea insertar. El sistema realiza la acción y finaliza el CU.	
Referencias	RF 3-7
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1- El diseñador selecciona la opción “Inserción” del menú.	1.1 El sistema despliega el menú.
2- El diseñador selecciona el tipo de modelo a insertar (árbol, señal, techo).	
3- El diseñador selecciona la posición(x,y,z) del nuevo modelo.	3.1 Verifica que la posición este dentro del entorno.
	3.2 Si la posición es correcta se crea una malla.
	3.3 El sistema busca la imagen a aplicar como textura.
	3.4 El sistema texturiza la malla con la imagen.
	3.5 Se visualiza en la posición indicada.

Curso alterno de los eventos	
Acción 3.2	Si la posición no es correcta se muestra un mensaje indicando al usuario retornar a la acción 3.
Precondiciones	Debe existir un entorno y las imágenes prediseñadas deben haber sido cargadas.
Poscondiciones	Se visualiza en nuevo modelos en la posición seleccionada.
Prioridad	Secundario

Tabla 8: CU Insertar modelo.

Nombre del caso de uso	**Cambiar propiedades.**
Actores	Diseñador
Propósito	Permite al diseñador cambiar color de fondo o color de fuente de letra, cambiar de la posición la cámara.
Resumen Se inicia cuando el Diseñador solicita del menú “Ver” la opción (Fondo, Fuente, Mover izquierda, Mover derecha, Mover arriba, Mover abajo, Acercar, Alejar). El sistema realiza la acción seleccionada y finaliza el caso de uso.	
Referencias	RF 20 - 29
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1- El diseñador selecciona la opción “Ver” del menú.	1.1 El sistema despliega el menú.
Escenario 1: Fondo	
2- El diseñador selecciona la opción “Fondo”	2.1 El sistema incrementa uno en la lista de colores.
	2.2 El sistema aplica el nuevo color de fondo.
Escenario 2: Fuente	
1- El diseñador selecciona la opción “Fuente”	2.1 El sistema incrementa uno en la lista de colores.

	2.2 El sistema aplica el nuevo color de fuente.
Escenario 3: Mover izquierda	
1- El diseñador selecciona la opción “Mover izquierda”	2.1 El sistema incrementa la posición de la cámara en el eje x.
Escenario 4: Mover derecha	
1- El diseñador selecciona la opción “Mover derecha”	2.1 El sistema decrementa la posición de la cámara en el eje x.
Escenario 5: Mover arriba	
1- El diseñador selecciona la opción “Mover arriba”	2.1 El sistema incrementa la posición de la cámara en el eje y.
Escenario 6: Mover abajo	
1- El diseñador selecciona la opción “Mover abajo”	2.1 El sistema decrementa la posición de la cámara en el eje y.
Escenario 7: Acercar	
1- El diseñador selecciona la opción “Acercar”	2.1 El sistema traslada la cámara a la posición seleccionada.
	2.2 El sistema incrementa la posición de la cámara en el eje z
Escenario 8: Alejar	
1- El diseñador selecciona la opción “Alejar”	2.1 El sistema decrementa la posición de la cámara en el eje z.
Precondiciones	Debe existir un entorno.
Poscondiciones	Es cambiada la posición de la cámara o el color de fondo o letra.
Prioridad	Opcional

Tabla 9: CU Cambiar propiedades.

Conclusiones

En el presente capítulo se definió qué es exactamente lo que espera el usuario con ésta herramienta. Para ello quedaron establecidos los requisitos funcionales de ésta, y descritos los casos de uso que le permitirá al cliente obtener los resultados esperados.

4

Capítulo 4 Análisis y diseño del Sistema

Introducción

La primera parte del capítulo encierra los estándares de codificación y a continuación los diagramas de clases de diseño y de secuencia por caso de uso como resultado del refinamiento de las etapas anteriores.

4.1 Estándares de codificación

El código de la herramienta 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.)). Se programará en inglés, debido 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 de la misma que cualquier módulo que se añada debe estar codificado siguiendo estos estándares.

Nombre de los ficheros:

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

STKNameOfUnits.cpp

Se usará **STK** para identificar el nombre de la herramienta (en definición!!)

Constantes:

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

Tipos de datos:

Los tipos se nombrarán siguiendo el siguiente patrón:

Enumerados: enum **E**MyEnum {**ME**_VALUE, **ME**_OTHER_VALUE};

Indicando con "**E**" que es de tipo enumerado. Nótese que las primeras letras de las constantes de enumerados son las iniciales del nombre del enumerado. Véase otro ejemplo:

enum **EN**ode**T**ype {**NT**_GEOMETRYNODE,…};

Estructuras: struct **S**MyStruct {…};

Indicando con "**S**" que es una estructura. Las variables miembros de la estructura se nombrarán igual que en las clases, leer más adelante.

Clases: class **C**ClassName;

Indicando con "**C**" que es una clase

Interfaces: **I**MyInterface

Indicando con "**I**" que es una interfaz.

Listas e iteradores STD:

vector<> **T**Name**List**;

TNameList::iterator **T**Name**ListIter**;

map<> **T**Name**Map**;

TNameMap::iterator **T**Name**MapIter**;

multimap<> **T**Name**MultiMap**;

TNameMultiMap::iterator **T**Name**MultiMapIter**;

Declaración de variables:

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

Tipos simples:

bool **b**VarName;

int **i**Name;

unsigned int **ui**Name;

float **f**Name;

char **c**Name;

char* **ac**Name; // arreglo de caracteres

char* **pc**Name; // puntero a un char

char** **aac**Name; // bidimensional

char** **apc**Name; // arreglo de punteros

bool **m_b**MemberVarName; //variable miembro

char **g*C***GlobalVarName; //variable global, no se le antepone “”

short **s**Name;

Instancias de tipos creados:

EMyEnumerated **e**Name;

SMyStructure **k**Name;

CClassName **k**ObjectName;

CClassName* **pk**Name; //puntero a objeto

CClassName* **ak**Name; //arreglo de objetos

CClassName* **ak**Name; // variable miembro de clase

IMyInterface* **pI**Name; //puntero interfaces

Métodos

En el caso de los métodos, se les antepondrá el identificador del tipo de dato de devolución, y en caso de no tenerlo (void), no se les antepondrá nada. Solamente los constructores y destructores comenzarán con "".En el caso de los argumentos se les antepone el prefijo "arg_"

Constructor y destructor:

CClassName (bool **arg**_bVarName, float& **arg**_fVarName);

~**C**ClassName ();

Funciones:

bool **b**Function1 (…);

int* **pi**Function2 (…);

CClassName* **pk**Function3 (…);

Procedimientos:

void Procedure4 (…);

Métodos de acceso a miembros

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

```
int iMyVar;  //variable
```

Obtención del valor:

```
int iMyVar();
{
   return iMyVar;
}
```

Establecimiento del valor:

```
void MyVar(char* arg_iMyVar)
{
  iMyVar = arg_iMyVar;
}
```

Obtención y establecimiento del valor:

```
int& iMyVar();
{
   return iMyVar;
}
```

4.2 Diagramas de clases de diseño

Por la complejidad del Diagrama de clases de diseño y para su mejor entendimiento se han agrupado algunas clases en paquetes; el paquete Scene ToolKit encapsula las clases que se utilizan de la herramienta básica y el paquete Map que encapsula las clases a implementar por la herramienta de creación de mapas. Las clases que se muestran de color blanco son las propias de SceneToolKit y las amarillas las de la herramienta de creación de mapas (Map).

Fig. 8: Diagrama de paquetes

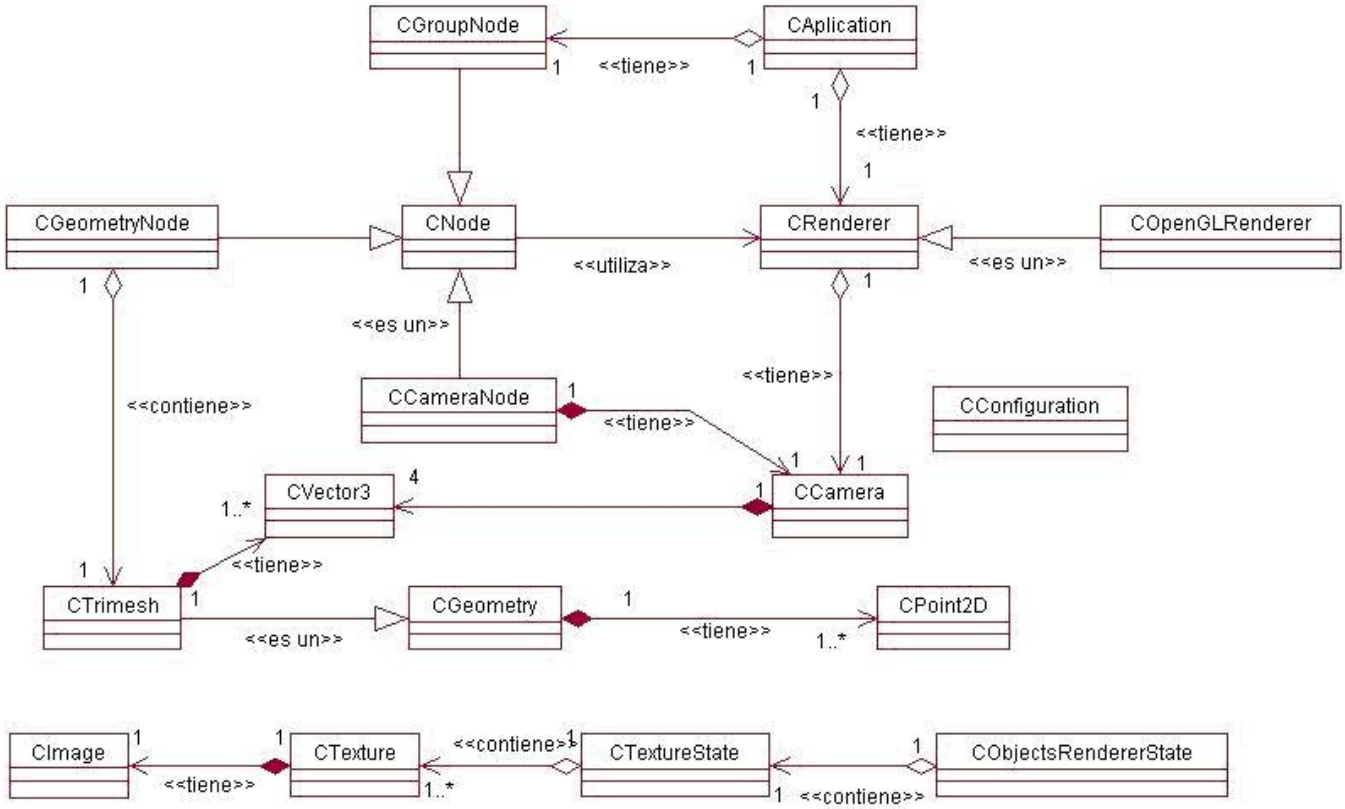

Fig. 9: Clases del paquete SceneToolKit

En la figura se muestran las clases del paquete SceneToolKit que será utilizado por la Herramienta de creación de mapas.

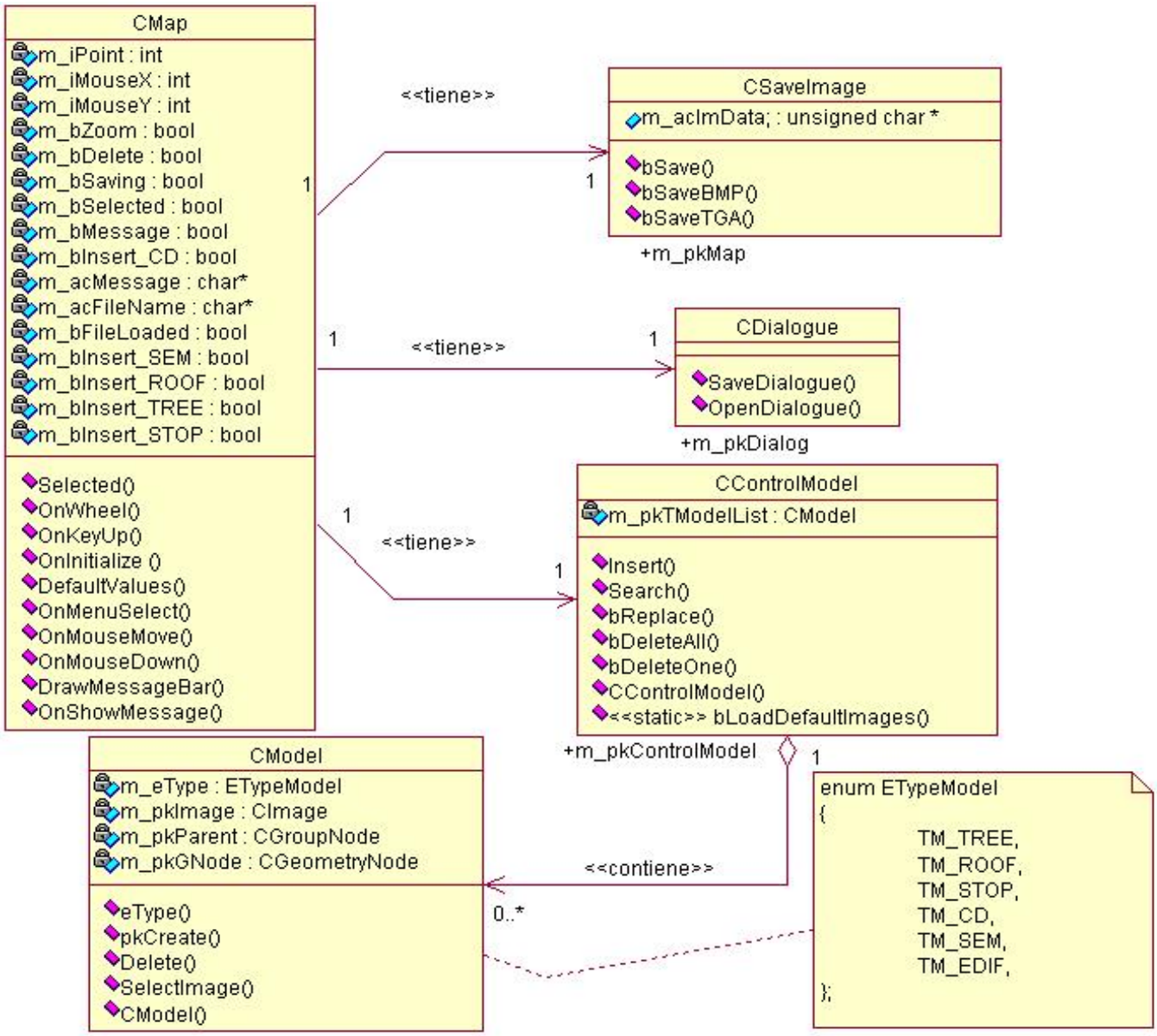

Fig. 10: Clases del paquete Map

En la figura se muestran las clases del paquete Map concebidas por la Herramienta de creación de mapas.

4.2.1 Descripción de las clases de diseño

Nombre: CMap	
Tipo de clase: Interfaz	
Atributo	**Tipo**
m_iPoint	int
m_iMouseX	int
m_iMouseY	int
m_bZoom	bool
m_bDelete	bool
m_bSaving	bool
m_bSelected	bool
m_bMessage	bool
m_bInsert_CD	bool
m_acMessage	char*
m_acFileName	char*
m_bFileLoaded	bool
m_bInsert_SEM	bool
m_bInsert_ROOF	bool
m_bInsert_TREE	bool
m_bInsert_STOP	bool
Operaciones	
Nombre:	Selected(int arg_iPosX, int arg_iPosY)
Descripción:	Función encargada de devolver el modelo seleccionado por el *mouse*, para los casos de eliminar donde el usuario selecciona el objeto que desea borrar.
Nombre:	OnWheel(short arg_sDelta)
Descripción:	Función encargada de acercar o alejar la cámara con el rodillo del *mouse.*
Nombre:	OnKeyUp(unsigned char arg_ucKey)
Descripción:	Función encargada de tomar el evento de cuando se levanta una tecla específica.

Nombre:	OnInitialize()
Descripción:	Función encargada de cargar las imágenes prediseñadas, ubicar la cámara de forma superior y guardar esa posición inicial.
Nombre:	DefaultValues()
Descripción:	Función encargada de inicializar los valores de las distintas variables.
Nombre:	OnMenuSelect(int arg_iItemID)
Descripción:	Función encargada de tomar la opción del menú que fue seleccionada.
Nombre:	OnMouseMove(int arg_iX, int arg_iY, unsigned int arg_uiModifiers)
Descripción:	Función encargada de trasladar la cámara con el *mouse*.
Nombre:	OnMouseDown(int arg_iButton, int arg_iState, int arg_iX, int arg_iY, unsigned int arg_uiModifiers)
Descripción:	Función encargada de tomar la posición donde el usuario de *click*.
Nombre:	DrawMessageBar()
Descripción:	Función encargada de visualizar el área donde se van a ubicar los mensajes.
Nombre:	OnShowMessage()
Descripción:	Función encargada de mostrar los mensajes.

Tabla 10:Descripción de la clase "CMap".

Nombre: CSaveImage	
Tipo de clase: Controladora	
Atributo	**Tipo**
m_acImData	unsigned char*
Operaciones	
Nombre:	bSave (char *arg_acFileName , int arg_iHeight , int arg_iWidth)
Descripción:	Función encargada de verificar la extensión con que se desea guardar la imagen y de acuerdo a esta llamar a los métodos bSaveBMP() o bSaveTGA().
Nombre:	bSaveBMP(char *arg_acFileName , int arg_iHeight , int arg_iWidth)
Descripción:	Función encargada convertir los datos del *buffer* de visualización a un fichero de extensión (.bmp).

Nombre:	bSaveTGA(char *arg_acFileName , int arg_iHeight , int arg_iWidth)
Descripción:	Función encargada convertir los datos del *buffer* de visualización a un fichero de extensión (.tga).

Tabla 11: Descripción de la clase "CSaveImage".

Nombre: CDialogue	
Tipo de clase: Controladora	
Operaciones	
Nombre:	SaveDialogue (HWND arg_iID, char* acLoadedFileName)
Descripción:	Función encargada de crear una ventana para tomar la dirección donde se desee salvar la imagen.
Nombre:	OpenDialogue (HWND arg_iID, char* acLoadedFileName)
Descripción:	Función encargada de crear una ventana para tomar la dirección desde donde se cargará el entorno.

Tabla 12: Descripción de la clase "CDialogue".

Nombre: CControlModel	
Tipo de clase: Controladora	
Atributo	**Tipo**
m_pkTModelList	Vector<CModel>*
Operaciones	
Nombre:	Insert (ETypeModel arg_pType, CVector3* arg_akVertex, CVector3 arg_kPosition)
Descripción:	Función encargada de crear un nuevo modelo de tipo arg_pType en la posición arg_kPosition.
Nombre:	Search()
Descripción:	Función encargada de hacer un pre-procesamiento de los modelos existentes para ganar en tiempo.
Nombre:	bReplace (ETypeModel arg_eType)
Descripción:	Función encargada de sustituir los modelos según el tipo definido por arg_eType.

Nombre:	bDeleteAll (ETypeModel arg_eType)
Descripción:	Función encargada de eliminar todos los modelos según el tipo definido por arg_eType.
Nombre:	DeleteOne (CNode* arg_pkNode)
Descripción:	Función encargada de eliminar un modelo seleccionado.
Nombre:	bLoadDefaultImages()
Descripción:	Función encargada de cargar las imágenes prediseñadas.

Tabla 13: Descripción de la clase "CControlModel".

Nombre: CModel	
Tipo de clase: Entidad	
Atributo	**Tipo**
m_eType	ETypeModel
m_pkImage	CImage
m_pkParent	CGroupNode
m_pkGNode	CGeometryNode
Operaciones:	
Nombre:	eType()
Descripción:	Función encargada de devolver el tipo de un CModel.
Nombre:	pkCreate (CVector3* arg_akVertex)
Descripción:	Función encargada de crear la malla y texturizarla con su imagen correspondiente.
Nombre:	Delete()
Descripción:	Función encargada de eliminar un CModel.
Nombre:	SelectImage (ETypeModel arg_eType, vector<CImage*>* arg_pkTImageList)
Descripción:	Función encargada de seleccionar la imagen según el tipo.
Nombre:	CModel (ETypeModel arg_eType, CGroupNode* arg_pkParent)
Descripción:	Constructor de la clase al cual se le pasa el tipo y el padre a quien se va a agregar.

Tabla 14: Descripción de la clase "CModel".

4.2.2 Caso de uso “Cargar entorno”

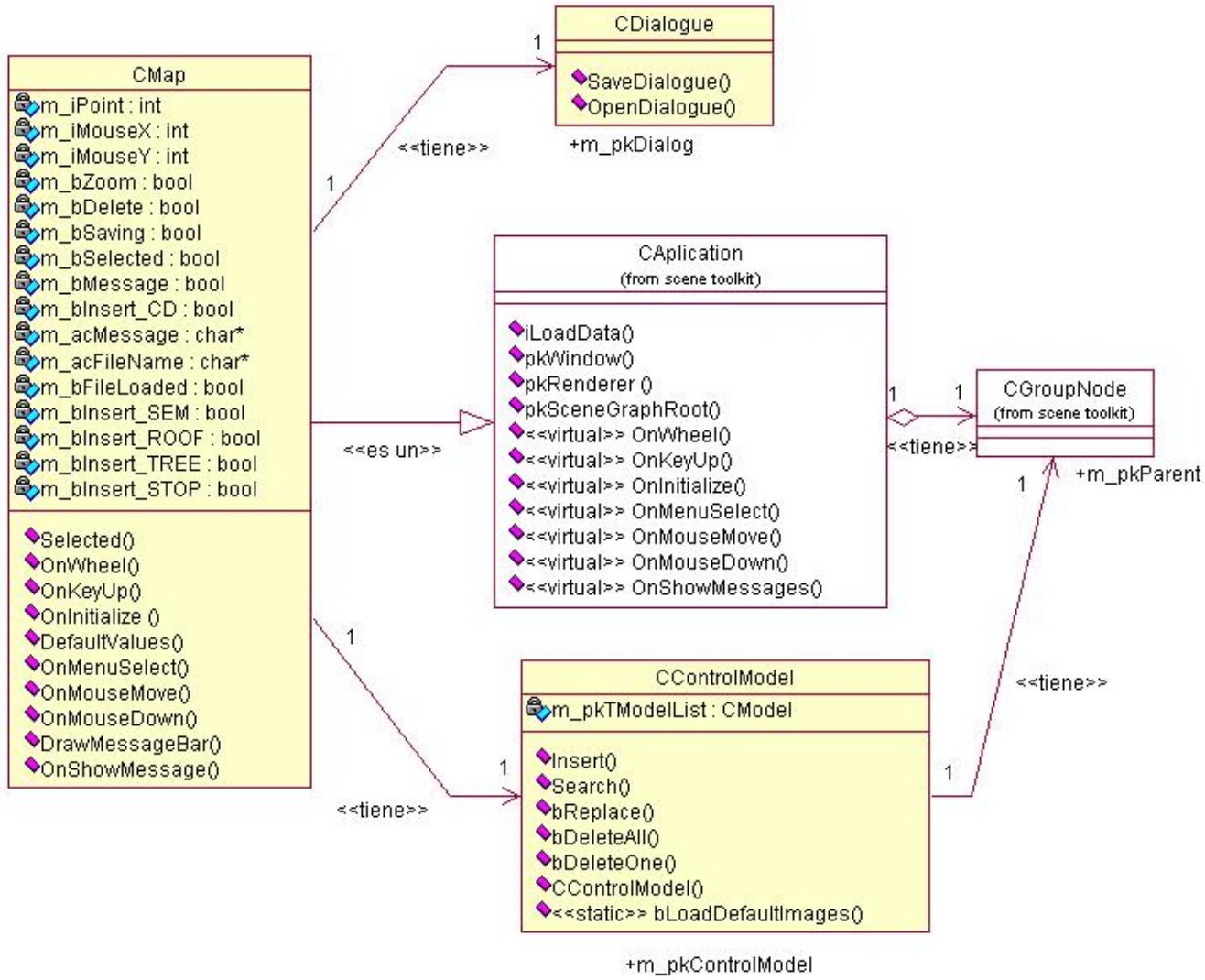

Fig. 11: Diagrama de clases de "Cargar entorno".

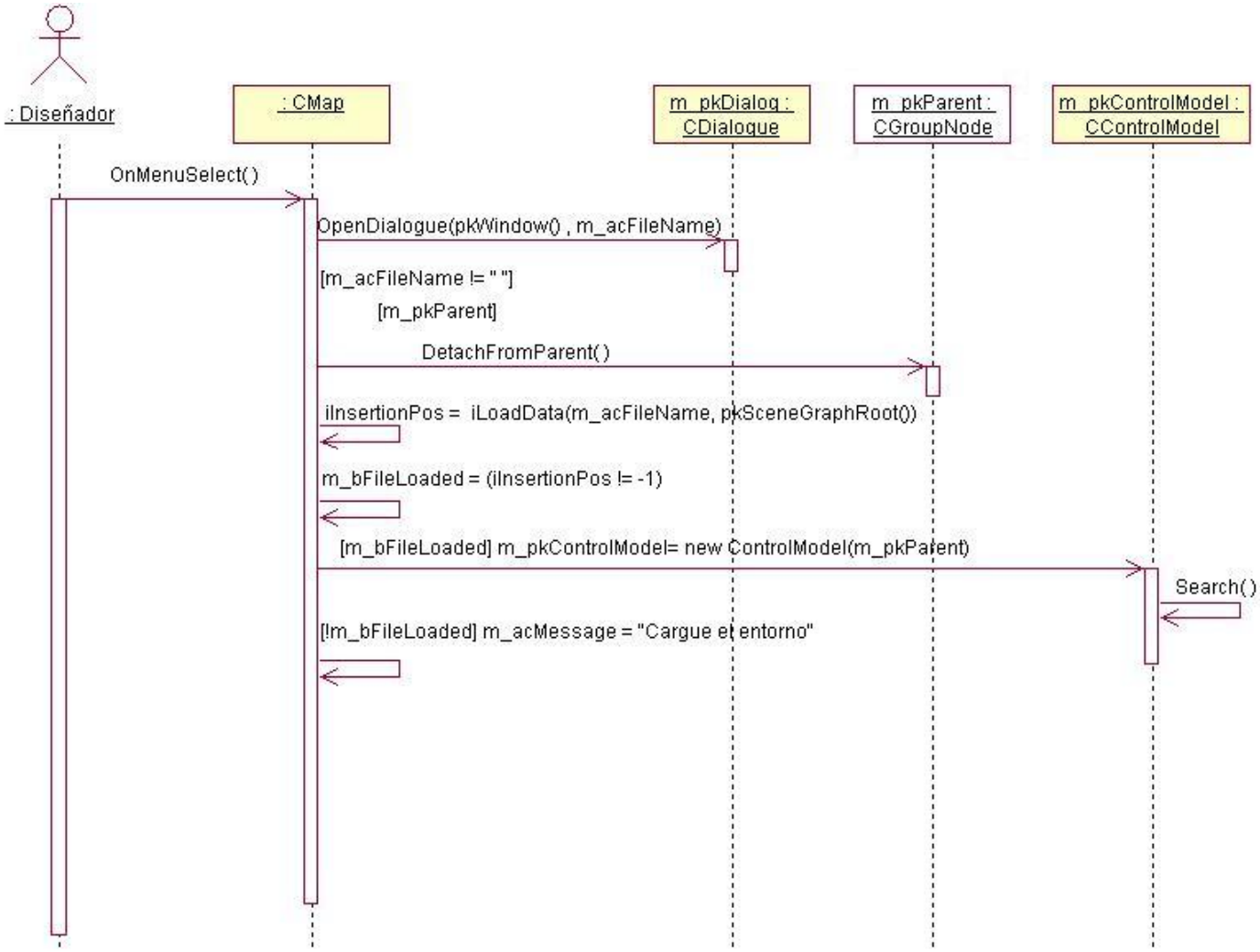

Fig. 12: Diagrama de secuencia de "Cargar entorno".

4.2.3 Caso de uso “Inicializar aplicación”

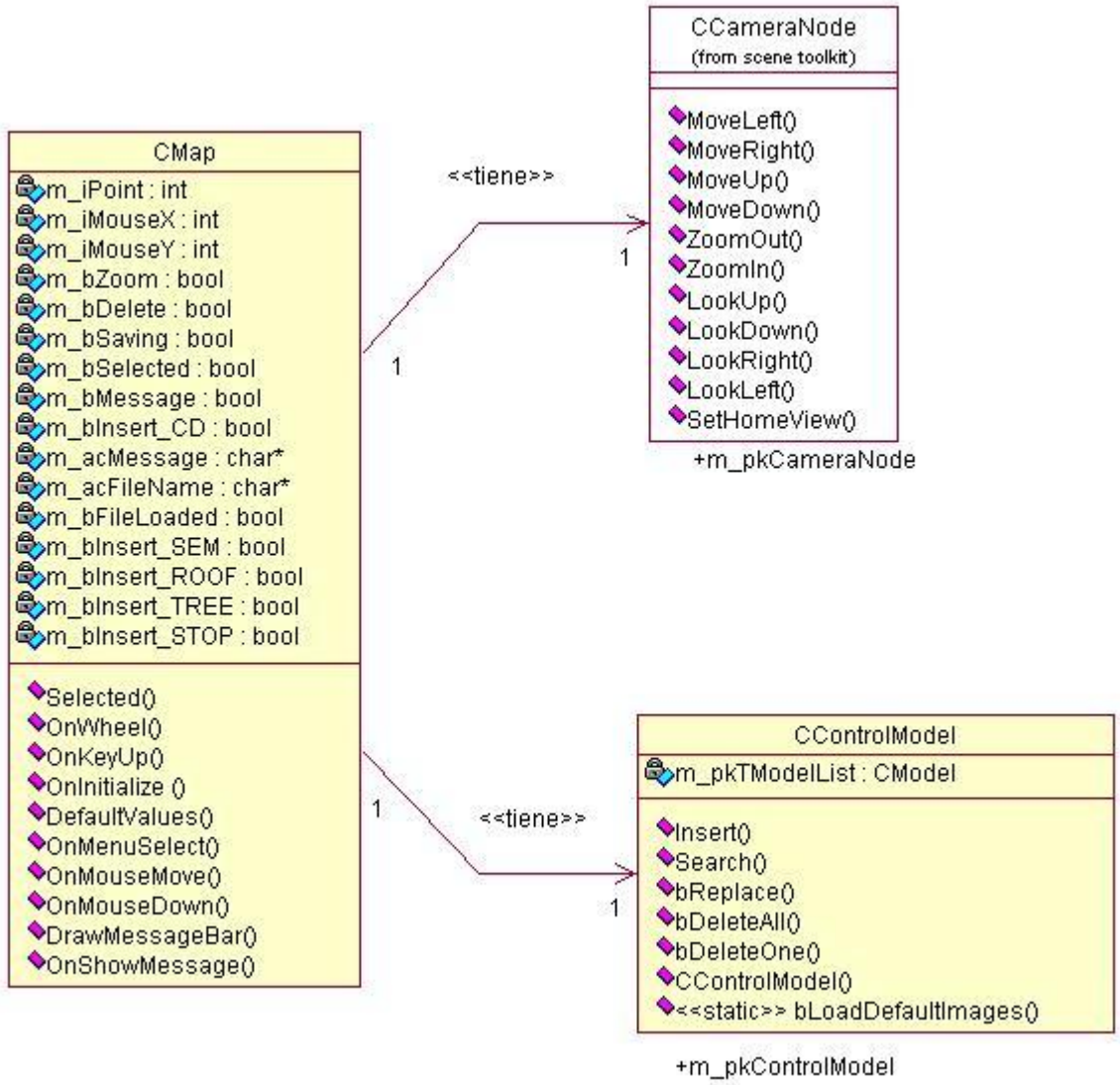

Fig. 13: Diagrama de clases de "Inicializar aplicación".

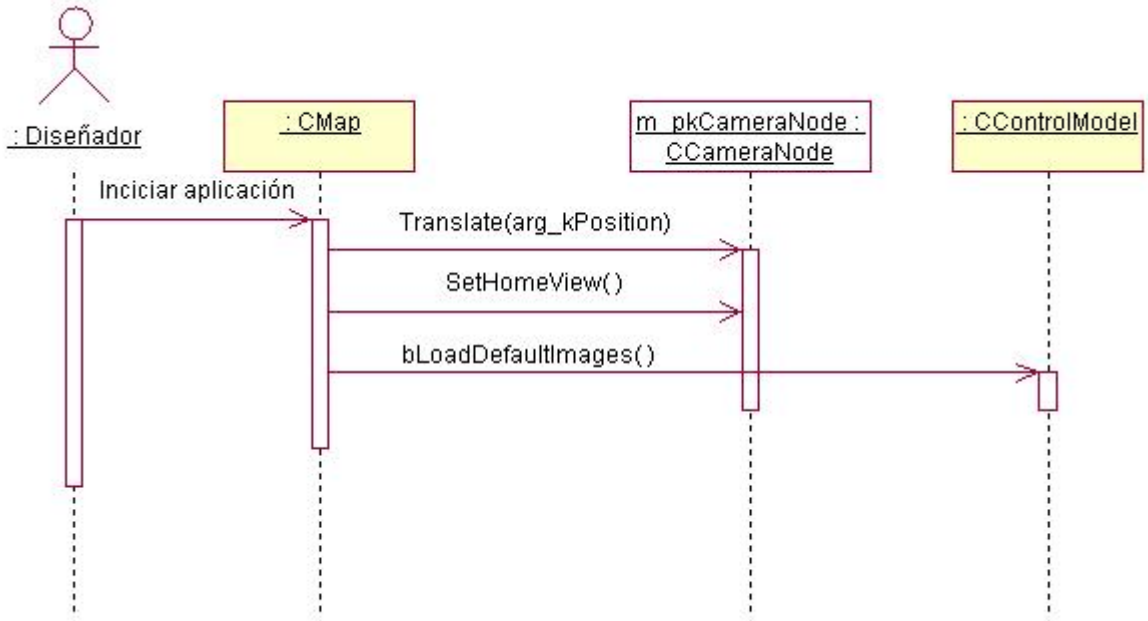

Fig. 14: Diagrama de secuencias "Inicializar aplicación".

4.2.4 Caso de uso “Salvar imagen”

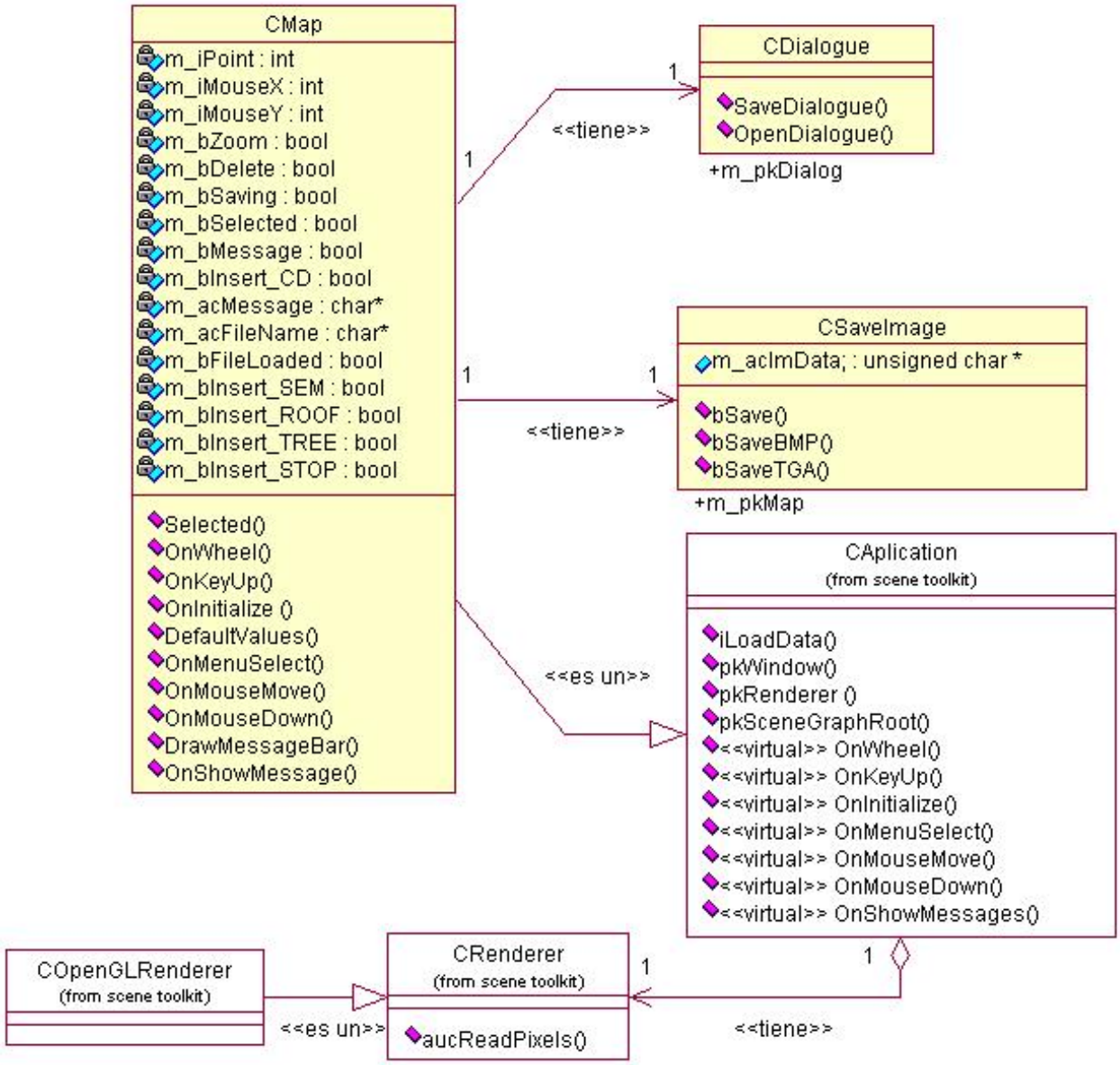

Fig. 15: Diagrama de clases "Salvar imagen"

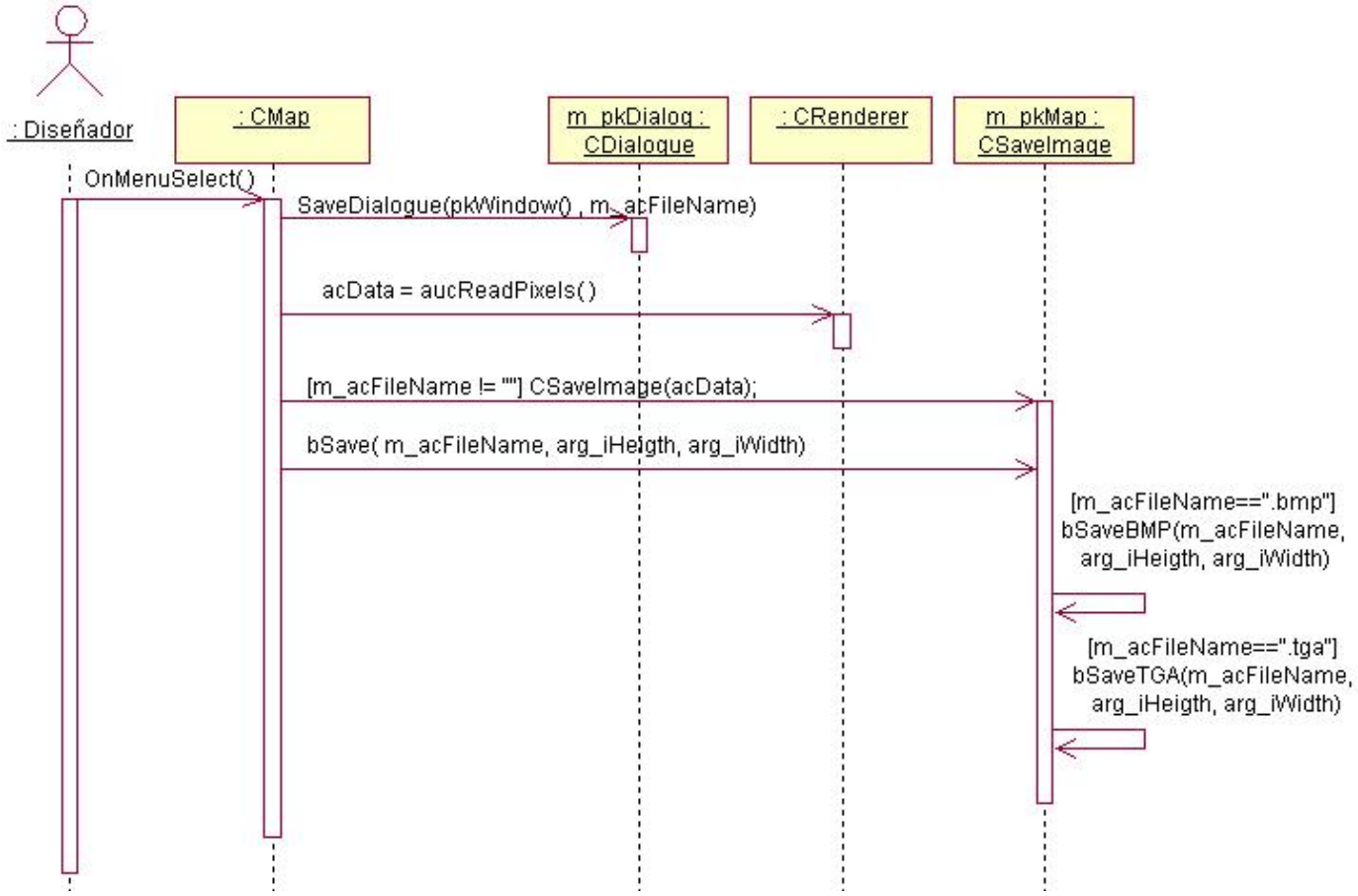

Fig. 16: Diagrama de secuencias "Salvar imagen".

4.2.5 Caso de uso “Sustituir modelo”

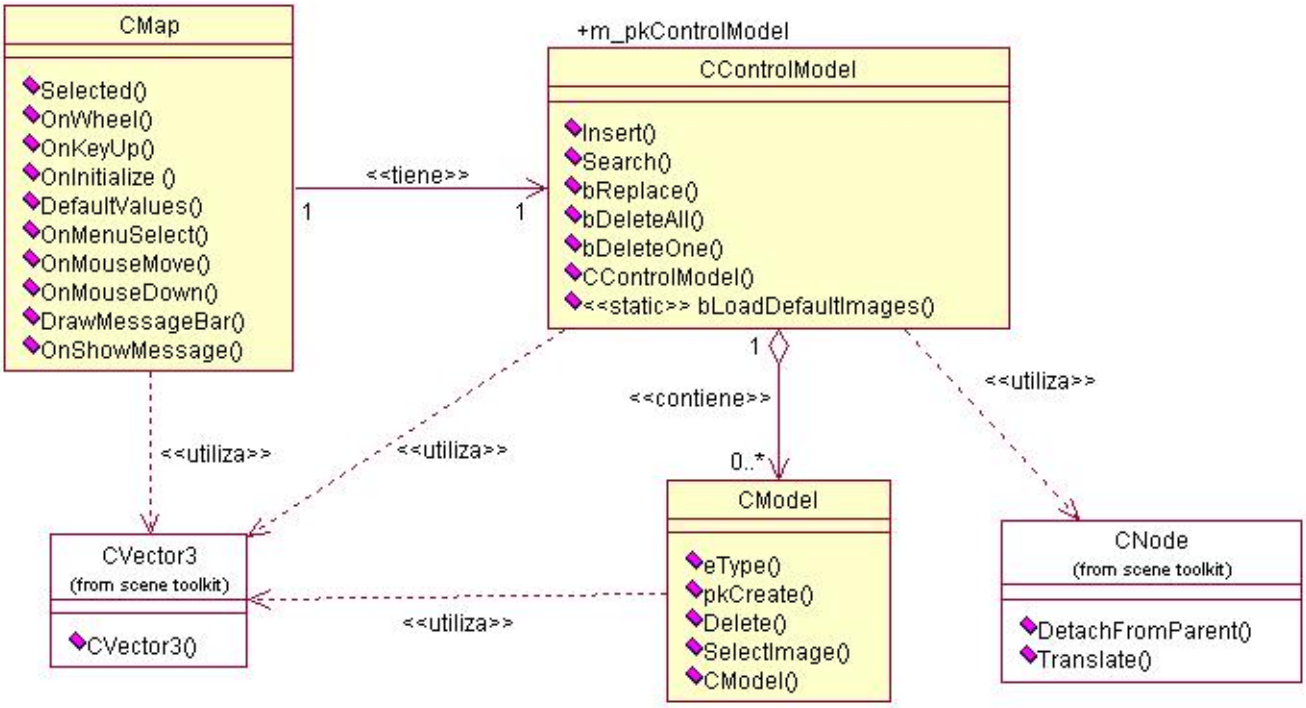

Fig. 17: Diagrama de clases "Sustituir modelo".

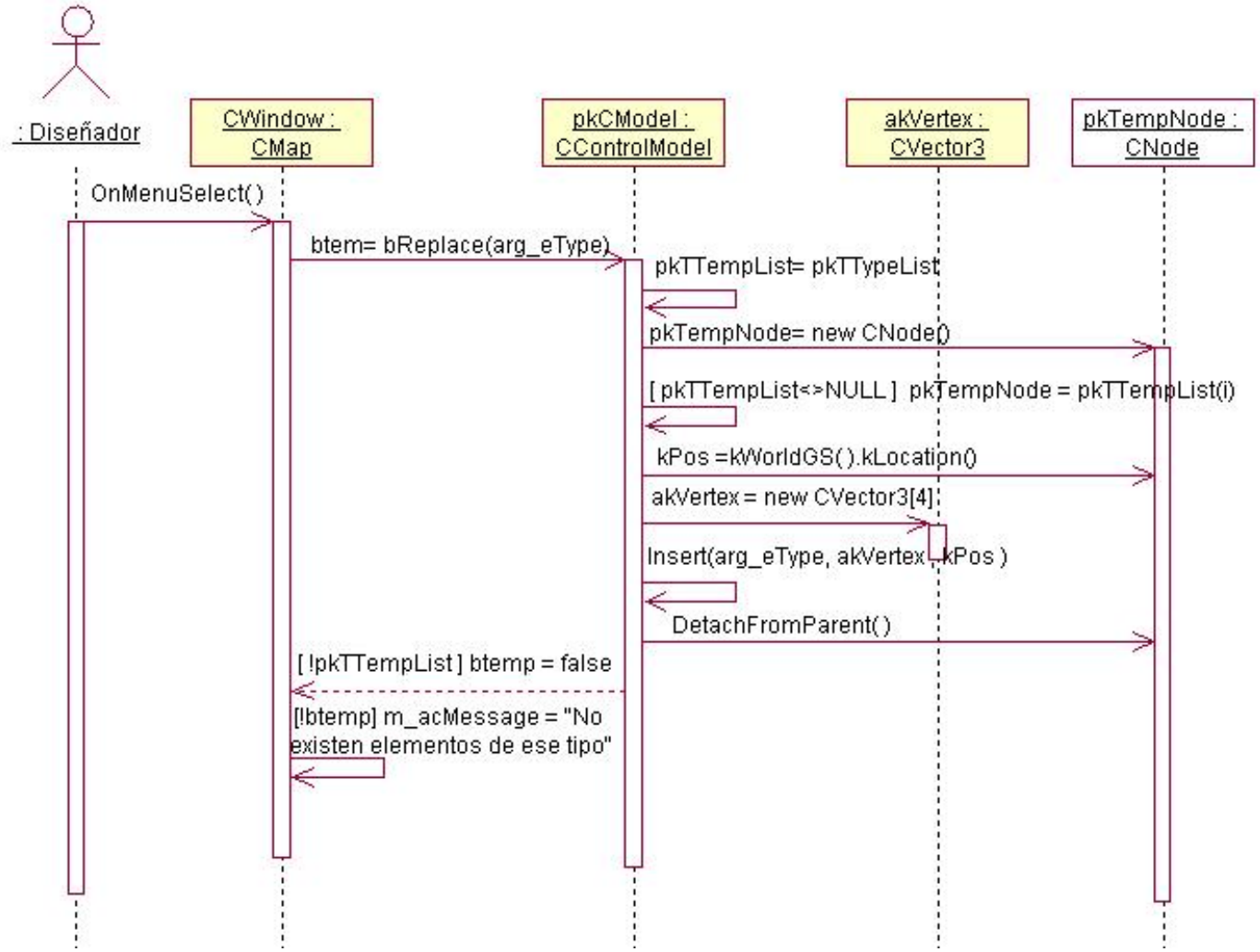

Fig. 18: Diagrama de secuencias "Sustituir modelo".

4.2.6 Caso de uso “Eliminar modelo”

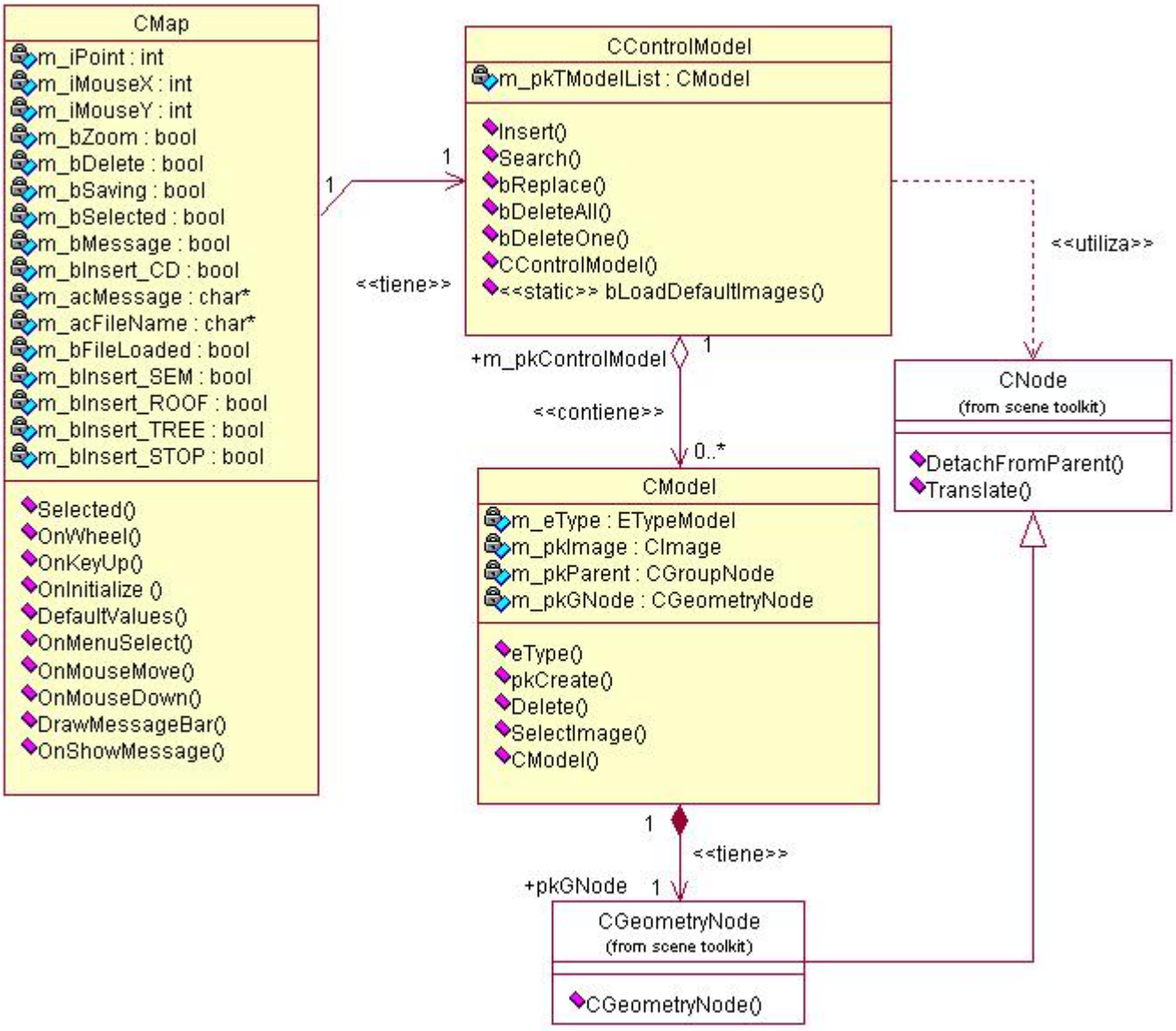

Fig. 19: Diagrama de clases "Eliminar modelo".

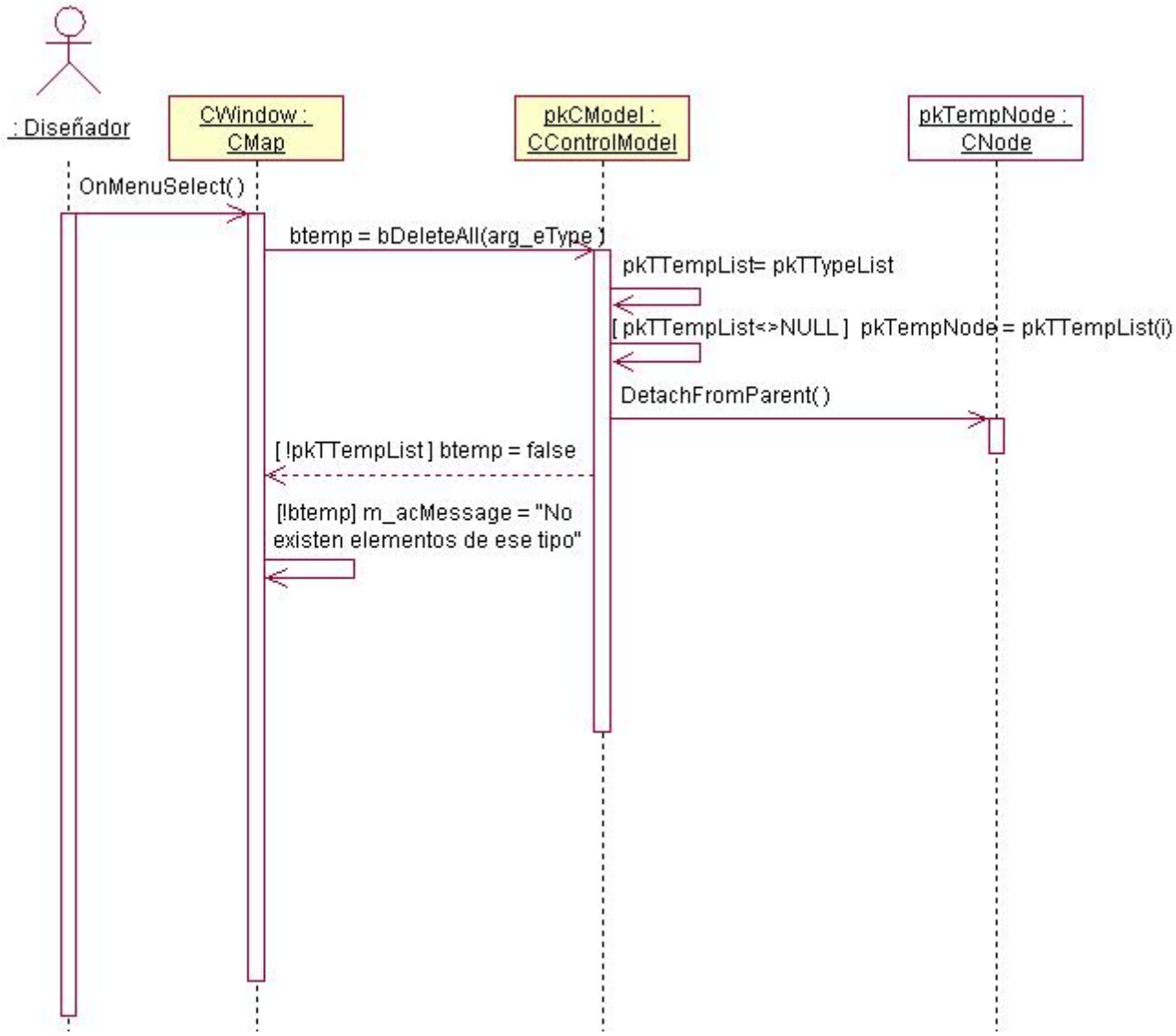

Fig. 20: Diagrama de secuencias "Eliminar modelo" (Todos reales).

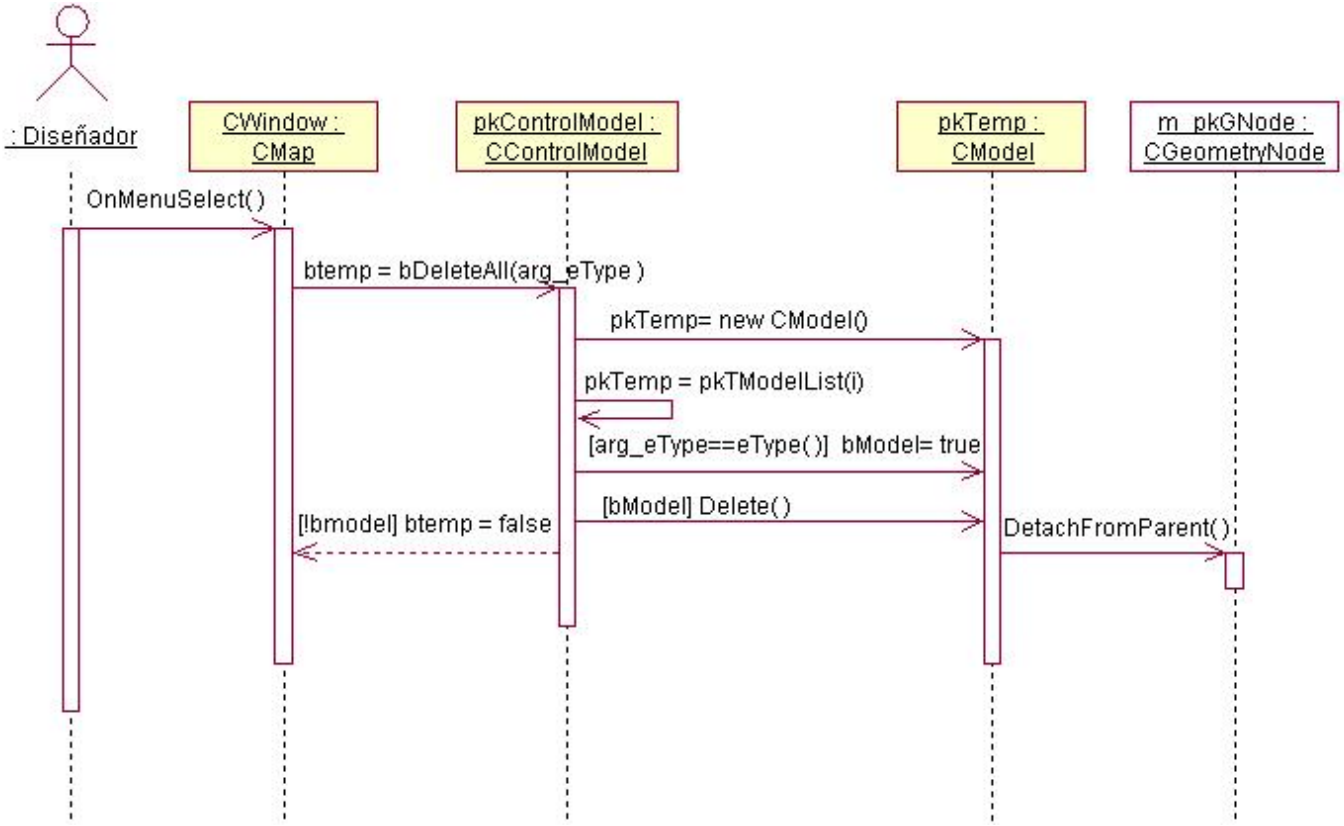

Fig. 21: Diagrama de secuencias "Eliminar modelo" (Todos sustitutos).

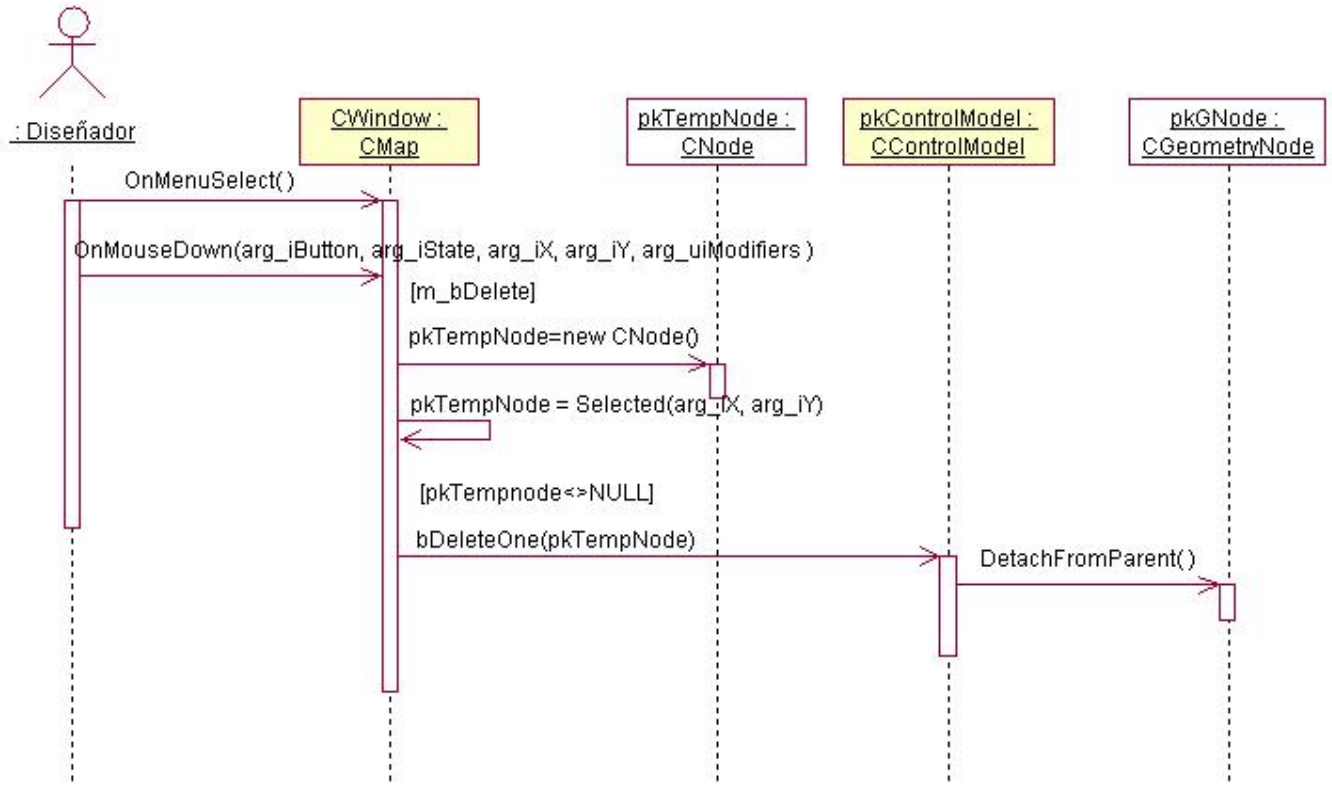

Fig. 22: Diagrama de secuencias "Eliminar modelo" (Solo uno).

4.2.7 Caso de uso “Insertar modelo”

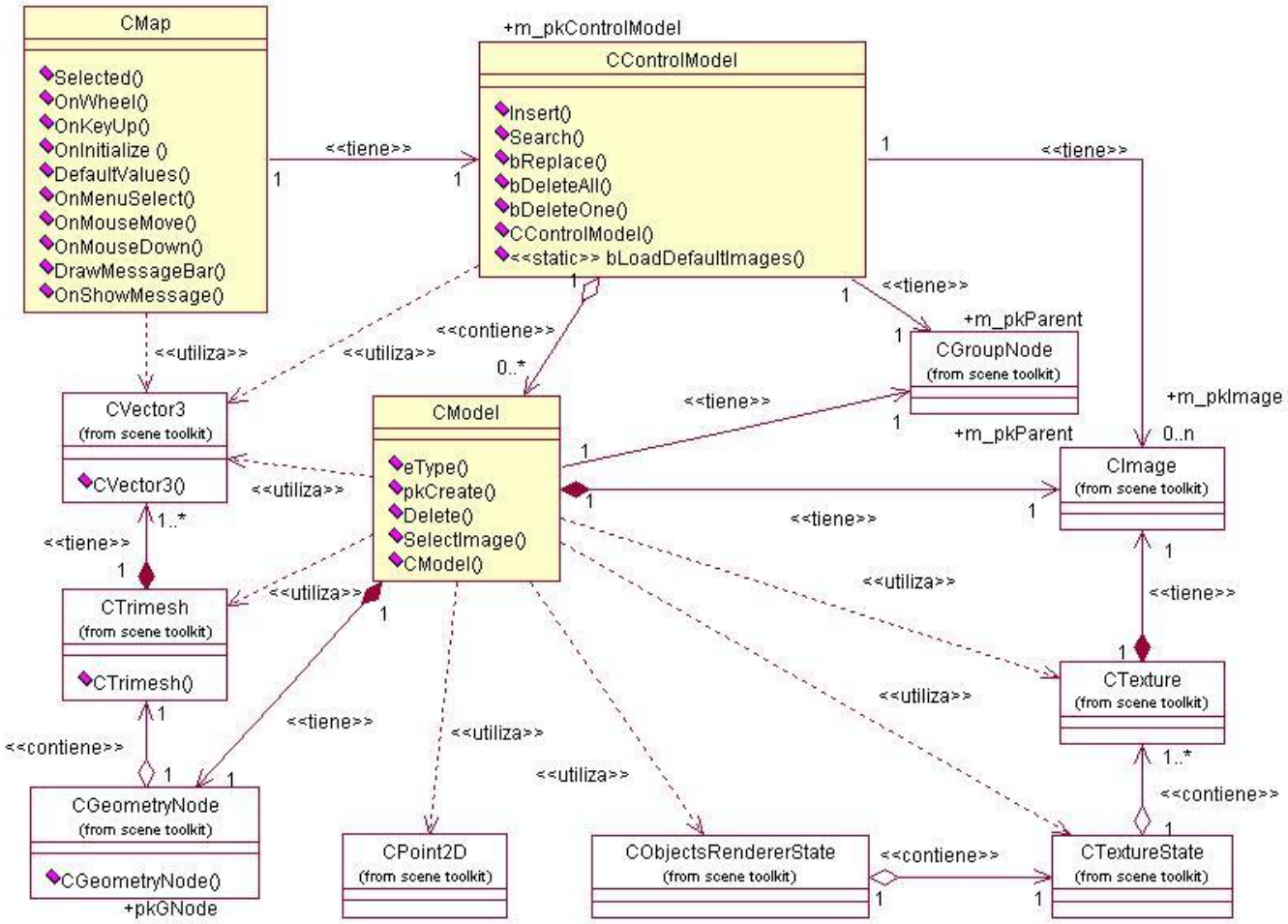

Fig. 23: Diagrama de clases "Insertar modelo".

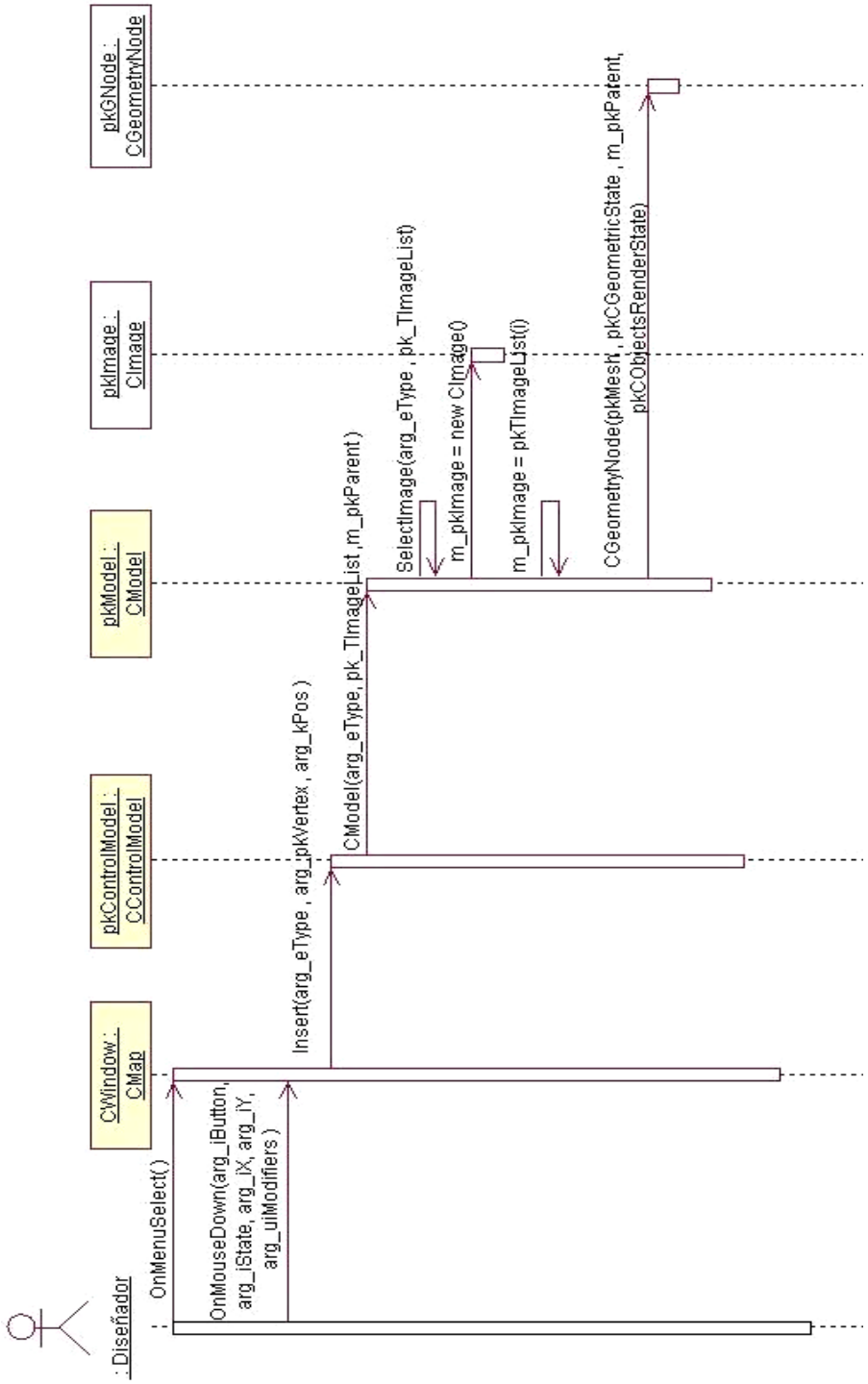

Fig. 24: Diagrama de secuencias "Insertar modelo".

4.2.8 Caso de uso “Cambiar propiedades”

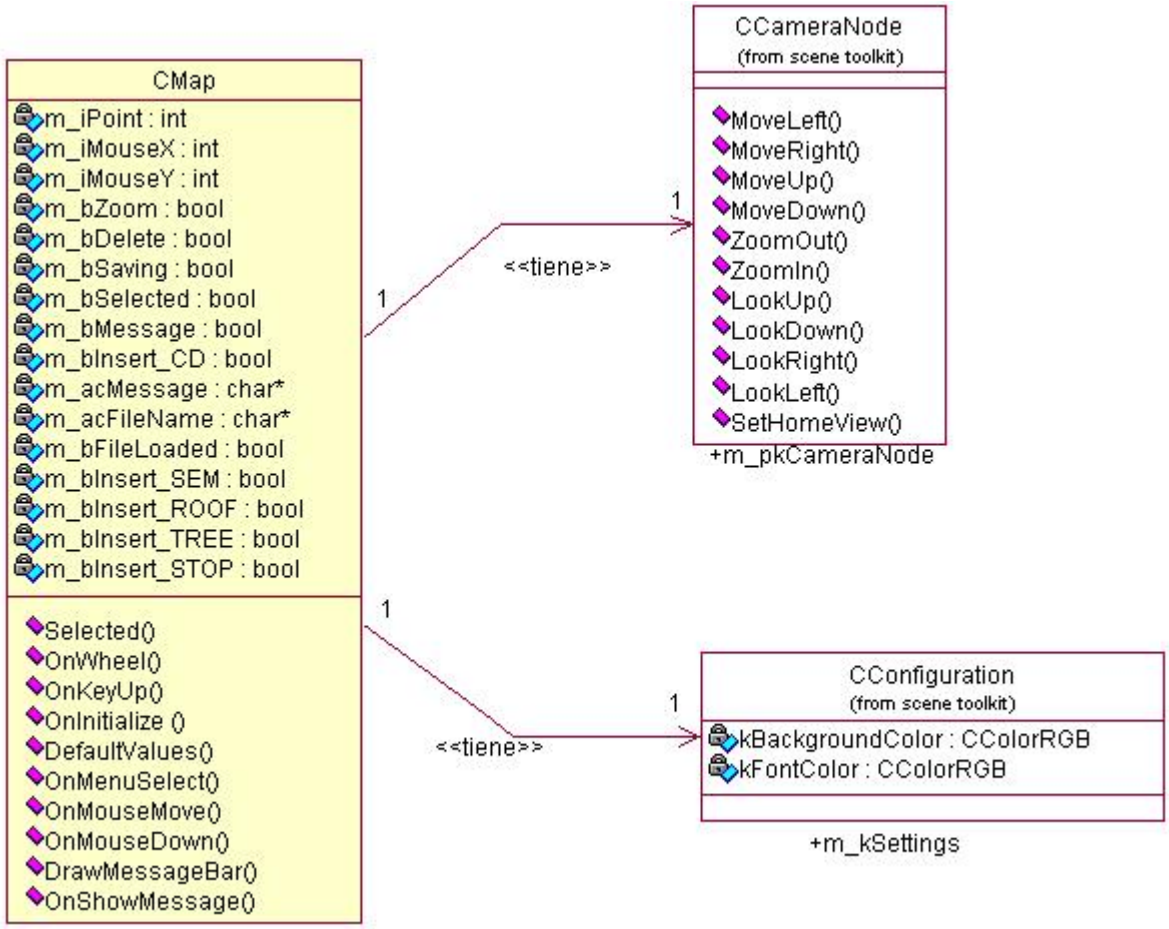

Fig. 25: Diagrama de clases "Cambiar propiedades".

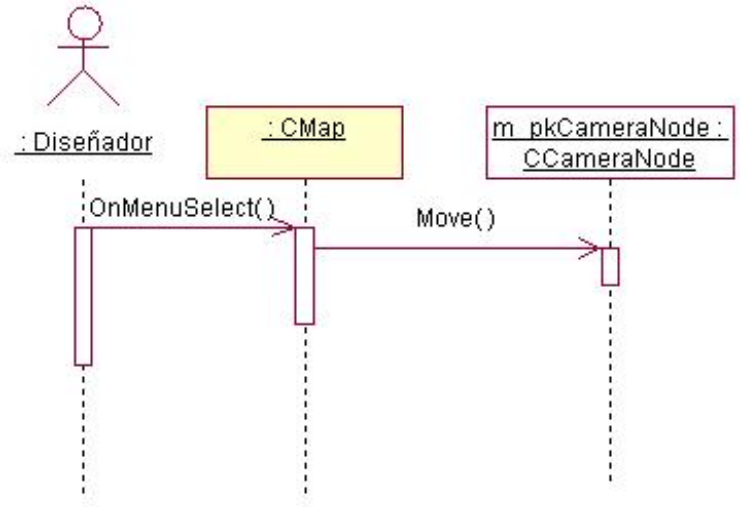

Fig. 26: Diagrama de secuencias "Cambiar propiedades" (Cámara).

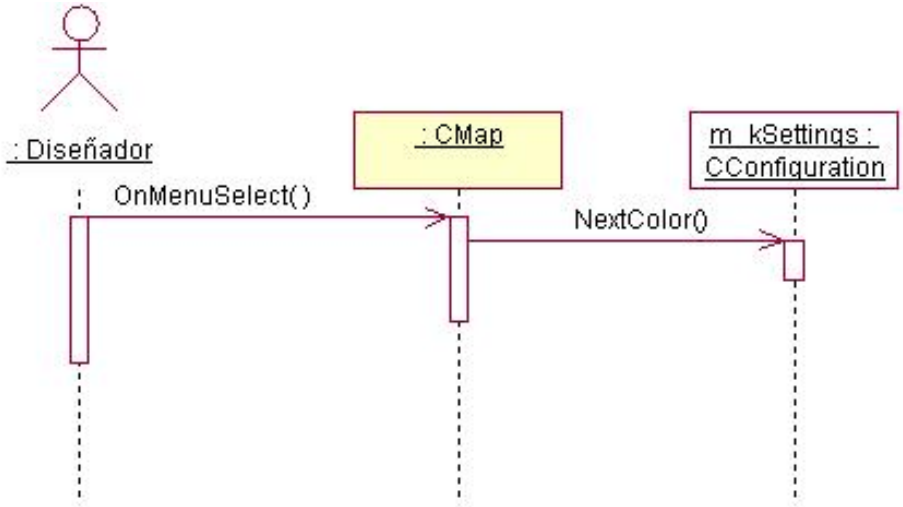

Fig. 27: Diagrama de secuencias "Cambiar propiedades" (Color).

Conclusiones

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

5

Capítulo 5 Implementación del Sistema

Introducción

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

5.1 Diagramas de components

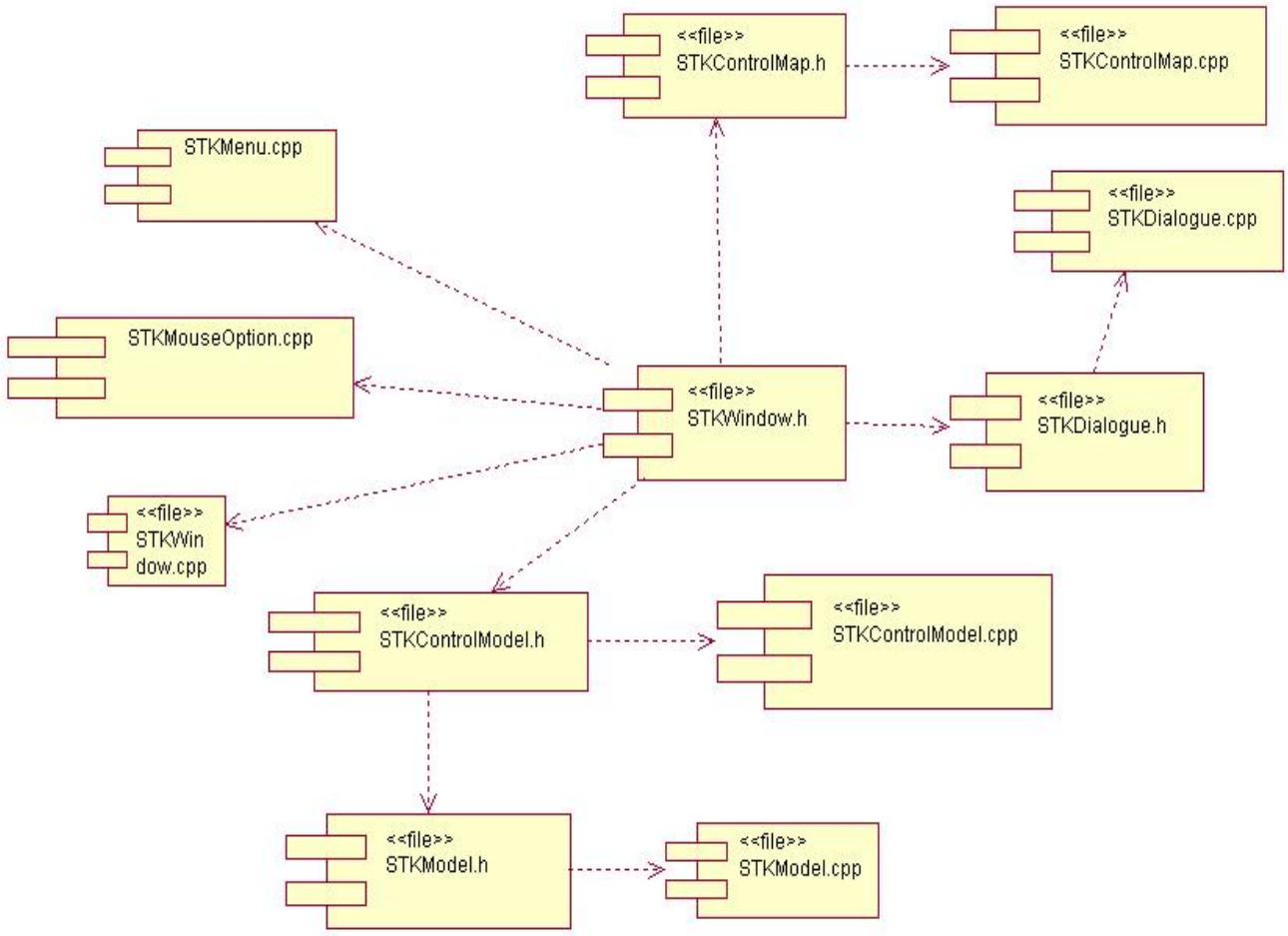

Fig. 28: Diagrama de componentes.

Conclusiones

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

Conclusiones

Para el cumplimiento de los objetivos de este proyecto, se realizó primeramente un estudio de las técnicas, tecnologías y tendencias actuales en relación con el tema. En el estudio se analizaron las características para desarrollar este tipo de aplicaciones. Además, a partir de esa investigación se proponen las características técnicas de la solución.

Posteriormente se realizó la captura de los requisitos funcionales y no funcionales, de la agrupación de dichos requisitos surgieron los primeros casos de uso del sistema, donde se realizó una descripción de cada uno.

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

Recomendaciones

Para que la Herramienta de creación de mapas brinde mayores facilidades de uso y servicios se recomienda:

- Implementar la captura de la imagen desde Directx.
- Implementar la funcionalidad que permite deshacer acciones.
- Incluir la reubicación con el *mouse* de un modelo ya creado.
- Permitir la definición de nuevos modelos sustitutos indicando los prefijos y las imágenes sustitutas.

Referencias bibliográficas

Libros

1. ASTLE, Dave y HAWKING, Kevin. *Capítulo 1: The exploration begins: OpenGL and DirectX*. En *OpenGL Game Programming*. USA, Prima Tech Publishing. 2001.

2. ASTLE, Dave y HAWKING, Kevin. *Capítulo 2: Using Windows with OpenGL*. En *OpenGL Game Programming*. USA, Prima Tech Publishing. 2001.

3. LUNA, Frank D. *Capítulo 6: Texturing.* En *Introduction to 3D Game Programming with DirectX*. USA, WordWare Publishing, Inc, 2003.

Libros digitales

4. IZNAGA, Arsenio PEREZ, Ivan. *Capítulo 1: Entidades geométricas.* En *Fundamentos de la Gráfica por Computadoras*.

5. IZNAGA, Arsenio PEREZ, Ivan. *Capítulo 3: Proyección.* En *Fundamentos de la Gráfica por Computadoras*.

6. CHOVER, Miguel. *Capítulo 2. Modelado geométrico I.* En *Introducción a la Informática Gráfica*.

7. CHOVER, Miguel. *Capítulo 3. Transformaciones.* En *Introducción a la Informática Gráfica*.

8. CHOVER, Miguel. *Capítulo 8. Modelado geométrico II.* En *Introducción a la Informática Gráfica*.

9. CHOVER, Miguel. *Capítulo 9. Aplicaciones de la Informática Gráfica.* En *Introducción a la Informática Gráfica*.

10. NEIDER Jackie DAVIS Tom WOO Mason. *Capítulo 1. Introduction to OpenGL.* En *OpenGL Programming Guide*. Addison-Wesley Publishing.

Sitios web

11. Gráficos 3D por computadora [Disponible en: http://es.wikipedia.org/wiki/Gr%C3%A1ficos_3D_por_computadora]

12. Matrices y Transformaciones [Disponible en: http://idam.ladei.com.ar/Tutoriales/Direct3D/Intro_MatTransf.html]

13. Representaciones de la superficie terrestre [Disponible en: http://www.edufuturo.com/educacion.php?c=503]

Índice de figuras y tablas

Índice de figuras

Índice de tablas

Apéndices

Glosario de abreviaturas

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

EG: Estado Geométrico.

ER: Estados de *Render*.

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

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

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

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

HW: Hardware.

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

SRV: Sistemas de Realidad Virtual.

SW: Software.

WGL: Wiggle.

Glosario de términos

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

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

Árbol: simbología que indica que en una posición dada hay un árbol.

Array: arreglo, orden, ordenación; colección, serie; vestido; (inform.) vector de datos, tipo de estructura de datos.

Entorno3D: fichero 3DX que será cargado por la aplicación para sustituir luego algunos de sus modelos.

Entorno sintético: mundo virtual.

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

ElementosVisualesSustitutos: Conjunto de elementos que tiene la aplicación que sustituirán algunos modelos del Entorno3D estos pueden ser señales de tránsito, árboles o techos.

Imagen: imagen que tendrán los ElementosVisulaesSustituos que representan las diferentes simbologías.

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, modelo geométrico que tienen los ElementosVisualesSustitutos.

Mapa: fichero con la información del plano resultante que es generado por la aplicación.

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

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

Modelo real: modelo geométrico que se encuentra en el entorno.

Modelo sustituto: modelo geométrico que suplantara al modelo real.

Normal: (ver vector normal).

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

Posición Raster: coordenadas de la pantalla donde se dibujará.

PropiedadesVisuales: algunas configuraciones que tiene la aplicación para facilitar el trabajo.

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

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

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

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

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

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

Renderizar: (ver *rendering*)

Señal: Simbología que indica que en una posición dada hay una señal de tránsito.

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

Techo: simbología que indica que en una posición dada hay un techo.

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

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

Texturizar: aplicar de texturas.

Tubería de rendering: (ver *Rendering pipeline*)

Vector: cantidad que expresa magnitud y dirección.

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

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

Vector unidad: vector de longitud 1.

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al Proyecto Herramientas de Desarrollo para Sistemas de Realidad Virtual de la Facultad 5 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

_____ _____

Yuremis Mengana Claro Yanoski Camacho Román

Firma del Autor Firma del Tutor

OPINIÓN DEL TUTOR DEL TRABAJO DE DIPLOMA

Título: Herramienta para la creación de mapas a partir de un entorno 3D.
Autor: Yuremis Mengana Claro

El tutor del presente Trabajo de Diploma considera que durante su ejecución el estudiante mostró las cualidades que a continuación se detallan.

Por todo lo anteriormente expresado considero que el estudiante está apto para ejercer como Ingeniero Informático; y propongo que se le otorgue al Trabajo de Diploma la calificación de:

_____	_____
Yanoski Camacho Román	
Firma	Fecha

DATOS DE CONTACTO

Nombre y Apellidos: Yanoski Rogelio Camacho Román
Edad: 26 años
Ciudadanía: cubano
Institución: Universidad de las Ciencias Informáticas (UCI)
Título: Ingeniero en Informática
Categoría Docente: Profesor Instructor
E-mail: rcamacho@uci.cu

Graduado de la CUJAE, con tres años de experiencia en el tema de la Gráfica Computacional, y líder de un proyecto de Realidad Virtual en la Universidad de las Ciencias Informáticas.