UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
CENTRO DE INFORMÁTICA INDUSTRIAL

Grafo de escena para la biblioteca GLSVe.

Tesis presentada para optar por el Título de Ingeniería en Ciencias Informáticas

Autor: **Yasmany Breff Pacheco**

Tutor: **MSc. Yoander Cabrera Díaz**

Co-tutor: **MSc. Liudmila Pupo Peña**

Ciudad de la Habana, fecha

DECLARACIÓN JURADA DE AUTORÍA

Declaro ser autor de la presente tesis y se le reconoce 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 _____.

_____

Firma del autor

Dedicatoria y agradecimientos

A mi madre: no me equivoco si digo que eres la mejor mamá del mundo, gracias por todo tu esfuerzo, tu apoyo y por la confianza que depositaste en mi. Gracias porque siempre, aunque lejos, has estado a mi lado. Te quiero mucho.

A mi padre: éste en un logro que quiero compartir contigo, gracias por ser mi papá y por creer en mi. Quiero que sepas que ocupas un lugar especial en mi.

A mi gran hermano: ya soy ingeniero igual que tú!!!!. Eres el mejor hermano tuve, tengo y tendré. Gracias por ser mi amigo, mi compañero y por ser parte de mi vida

A mi tía: me siento orgulloso de tener la dicha de poseer dos madres, mami, y tú. Para ti un beso y un abrazo de este hijo que te quiere mucho. Gracias por darme ese apoyo que tanto necesite en estos 5 años de carrera.

A mi novia: primeramente quiero agradecerle a la vida por haberme dado algo tan especial como tú. Quiero que sepas que eres lo mas lindo que me podría haber pasado. Has sido mi fiel compañera en la distancia que ha sabido comprenderme, quererme y darme el apoyo que necesito para llegar a ser lo que hoy soy. A ti mi princesa, Gracias por ser mi novia.

A todos mis amigos sin excluir a ninguno, los que me han acompañado siempre, mi negrito Loren, el Migue, Leo, Yixo, Albert, Yuvi, Lazaro, y a los que conocí aquí en la escuela Osmany, Guille, Leo, Fumero, Consu, Roiley, en fin a todos gracias por formar parte de mi vida.

A mis tutores: por guiarme en mi formación como profesional y brindarme la oportunidad de ser útil en lo que hago.

A todos Gracias.....

Breff

Resumen.

En este trabajo se desarrolla un grafo de escena como alternativa de estructura de datos jerárquica para una biblioteca de visión estereoscópica y realidad virtual. Esta biblioteca tiene distintos campos de aplicación, lo que hace necesario implementar diversas funcionalidades en el grafo de escena a fin de ofrecer mayor prestación a la misma. En primer lugar y como característica singular y más importante, la biblioteca es utilizada en aplicaciones de evaluación y entrenamiento de funciones visuales como la agudeza y la visión binocular.

El trabajo está centrado en la implementación de un grafo de escena que permita organizar lógicamente un escenario complejo. Dicho grafo está compuesto por distintos tipos de nodos que permiten al desarrollador tener un mejor control de los modelos geométricos dibujados en la escena.

Para dar cumplimiento a los objetivos de esta investigación se hará un estudio de las diferentes estructuras de datos usadas en diversos motores gráficos.

Esta propuesta es un componente esencial para la biblioteca a fin de mejorar el rendimiento en el manejo de información de la misma en la visualización de distintos tipos de aplicaciones, sin descartar que los aportes realizados por el autor a la herramienta contribuyen a que la misma posibilite una amplia utilización en diversas ramas de la ciencia, la docencia y la investigación.

Palabras clave:
biblioteca gráfica, estructura de datos, grafo de escena, nodo, visión estereoscópica.

Índice general

Introducción **11**

1. Fundamentación Teórica **14**

1.1. Estructura de datos 15

1.1.1. Lista enlazada 15

1.1.2. Pila 18

1.1.3. Cola 20

1.1.4. Árbol 22

1.1.5. Grafo 25

1.1.6. Grafo de Escena 27

1.2. Motores gráficos. Estructura de datos 29

1.2.1. Ogre3D 29

1.2.2. Delta3D 30

1.2.3. Unreal Engine 30

1.2.4. Wild Magic 30

1.2.5. SceneTollKit 30

1.3. Consideraciones generales 31

1.3.1. Listas 31

1.3.2. Pila 31

1.3.3. Cola 31

1.3.4. Árbol 32

1.3.5. Grafo 32

1.3.6. Grafo de escena 32

2. Solución Propuesta **33**

2.1. Descripción del problema 34

2.1.1. Justificación de la implementación de una estructura de datos jerárquica para la GLSVe 34

2.2. Graphics Library for Stereoscopic Vision (GLSVe) . . . 34
2.3. Selección de una Estructura de datos . . . 34
2.4. Grafo de Escena . . . 35
2.4.1. Estructura del Grafo de Escena . . . 35
2.4.2. Tipos de Nodos . . . 36
2.4.3. Operaciones que brinda el Grafo de Escena . . . 37
2.4.4. Incorporación del grafo de escena a la GLSVe . . . 43
2.5. Consideraciones generales . . . 44

3. Diseño de la solución 45
3.1. Modelo de Dominio . . . 46
3.2. Captura de Requisitos . . . 47
3.2.1. Requisitos Funcionales . . . 47
3.2.2. Requisitos no Funcionales . . . 48
3.3. Modelo de Casos de Uso del Sistema . . . 48
3.3.1. Actores de Sistema . . . 49
3.3.2. Diagrama de Caso de Uso del Sistema . . . 49
3.3.3. Descripción de Casos de Uso del Sistema . . . 50
3.4. Diagrama de Clases de Diseño . . . 55
3.5. Diagrama de Secuencia del Diseño . . . 56

4. Implementación y Análisis de resultados 61
4.1. Diagrama de componente . . . 62
4.2. Validación de los Resultados . . . 62
4.3. Datos de Prueba . . . 63
4.4. Casos de Prueba . . . 63
4.4.1. Funcionalidades incorporadas a la biblioteca GLSVe . . . 63
4.4.2. CP 1: Adicionar nodo . . . 64
4.4.3. CP 2: Eliminar nodo . . . 65
4.4.4. CP 3: Trasladar nodo . . . 65
4.4.5. CP 4: Rotar nodo . . . 66
4.4.6. CP 5: Escalar nodo . . . 67
4.4.7. Visualización de la actividad de Agudeza Visual por Transparencia 68
4.4.8. Visualización de otras escenas con el Grafo de escena . . . 70

Conclusiones 71

Recomendaciones **72**

Referencias bibliográficas **73**

Glosario de Términos **75**

Acrónimos **76**

Índice de figuras

1.1. *Estructuras de datos* 16
1.2. *Lista simplemente enlazada* 17
1.3. *Lista doblemente enlazada* 17
1.4. *Ejemplo de lista homogénea y heterogénea* 18
1.5. *TDA Pila* 18
1.6. *Ejemplo de funcionamiento de una Pila* 19
1.7. *Invirtiendo la cadena "casa" utilizando una pila* 20
1.8. *Representación de una cola* 21
1.9. *Ejemplo de funcionamiento de una cola* 21
1.10. *Árbol* 23
1.11. *Grafo* 25
1.12. *Grafo de escena* 28

2.1. *Grafo de GLSVe* 36
2.2. *Estado del grafo de escena luego de eliminar un nodo* 38
2.3. *Ejemplo del cambio de padre de un nodo* 40
2.4. *Estado de un nodo al ser aplicado el Traslate* 41
2.5. *Estado de un nodo al ser aplicado el Rotate* 41
2.6. *Estado de un nodo al aplicarle un LookAt* 42
2.7. *Estado del grafo luego de ser actualizado* 42
2.8. *Grafo de escena incorporado a la GLSVe* 43

3.1. *Modelo de dominio* 46
3.2. *Diagrama de Caso de Uso del Sistema* 49
3.3. *Diagrama de Clases de Diseño* 55
3.4. *Diagrama de Secuencia del CU Adicionar* 56
3.5. *Diagrama de Secuencia del CU Eliminar* 57
3.6. *Diagrama de Secuencia del CU Buscar* 58

3.7. *Diagrama de Secuencia del CU Dibujar* 59
3.8. *Diagrama de Secuencia del CU Modificar* 60

4.1. *Diagrama de Componentes* 62
4.2. *Actividad de transparencia cargada con la lista* 68
4.3. *Ejemplo de código con lista* 69
4.4. *Ejemplo de Actividad de transparencia cargada con el grafo de escena* . . . 69
4.5. *Ejemplo de código con grafo de escena* 70
4.6. *Ejemplo de escena cargada con el grafo de escena* 70
4.7. *Ejemplo de actividad de vision estereoscópica cargada con el grafo de escena* 70

Índice de tablas

2.1. *Tabla de características de Estructuras de datos* 34
2.2. *Tabla de tipos de nodos y sus descripciones* 37

3.1. *Actor del sistema* 49
3.2. *Descripción del CU Gestionar Nodos* 50
3.3. *Descripción de la sección Adicionar Nodo del CU Gestionar Nodo.* 51
3.4. *Descripción de la sección Eliminar Nodo del CU Gestionar Nodo.* 51
3.5. *Descripción del CU Buscar Nodo.* 52
3.6. *Descripción del CU Dibujar Nodo.* 53
3.7. *Descripción del CU Modificar Nodos.* 54

4.1. *Funcionalidades de GLSVe* 63
4.2. *Caso de prueba adicionar nodo* 64
4.3. *Especificación del caso de prueba adicionar nodo* 64
4.4. *Caso de prueba eliminar nodo* 65
4.5. *Especificación del caso de prueba eliminar nodo* 65
4.6. *Caso de prueba trasladar nodo* 65
4.7. *Especificación del caso de prueba trasladar nodo* 66
4.8. *Caso de prueba rotar nodo* 66
4.9. *Especificación del caso de prueba rotar nodo* 66
4.10. *Caso de prueba escalar nodo* 67
4.11. *Especificación del caso de prueba escalar nodo* 67

Introducción

La RV[1] es un sistema interactivo que permite sintetizar un mundo tridimensional ficticio, a través de gráficos 3D. Básicamente consiste en simular la mayor cantidad de percepciones de una persona: 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 entorno virtual generado por la computadora. La RV contiene ramas muy variadas de trabajo dentro de las que se encuentran: la visualización científica (física, médica, etc.), entretenimiento, apreciación de espacios arquitectónicos, diseño asistido por computadora, etc[Mon, 2010].

La gestión y manipulación de los elementos que se encuentran desplegados en el espacio tridimensional son hechos mediante una ED[2], la cual es uno de los componentes fundamentales de un motor gráfico.

Numerosos tipos de ED se han desarrollado, y algunas son muy generales y ampliamente utilizadas, mientras que otras son altamente especializadas para determinados tipos de tareas. La selección cuidadosa de las estructuras de datos puede permitir el uso de los algoritmos más eficientes para tareas específicas y por lo tanto optimizar el rendimiento de los programas[mis, 2006].

En el proyecto Herramientas de Desarrollo para Sistemas de Visión Estereoscópica HDSVE[3] se desarrolla la Biblioteca Gráfica de Visión Estereoscópica (Graphic Library for Stereoscopic Vision) GLSVe[4]. Esta biblioteca es usada en la creación de aplicaciones de VE[5]. La programación de estos sistemas se hace compleja a la hora de acceder a los objetos, esto se debe a que la biblioteca para la gestión de dichos objetos en la escena basa su funcionamiento en una ED denominada lista que no es óptima pues la manipulación y búsqueda se hace costosa y no establece jerarquía entre los mismos.

[1] Realidad Virtual
[2] Estructura de Datos
[3] Herramientas de Desarrollo para Sistemas de Visión Estereoscópica
[4] Graphics Library for Stereoscopic Vision engine
[5] Visión estereoscópica

A raíz de la problemática planteada se formula como **problema científico**: ¿Cómo mejorar el procesamiento de información en la construcción de escenas en la GLSVe?

El problema científico anterior, define que el **objeto de estudio** corresponde a: Las ED para mejorar del procesamiento de información en bibliotecas gráficas.

Con el propósito de brindar solución al problema, se plantea como **objetivo de investigación** diseñar e implementar una ED jerárquica capaz de mejorar el manejo de escenas complejas en la GLSVe.

El **campo de acción** se enfoca en las estructuras de datos jerárquicas para mejorar el procesamiento de información en la GLSVe.

Para lograr el total cumplimiento del objetivo se trazan las siguientes **tareas investigativas**:

- Estudio de aplicaciones existentes que poseen ED para la organización jerárquica de los elementos con que cuenta.
- Diseño de una arquitectura lógica e implementación de una solución en la GLSVe.
- Validación de la solución propuesta en la GLSVe, partiendo de la construcción de prototipos de actividades para la manipulación de elementos en escena.

Idea a defender: Con la implementación de una estructura de datos jerárquica se logrará hacer más sencillo el procesamiento y el acceso a los objetos en la escena.

Estructura del documento

El presente trabajo está estructurado por cuatro capítulos, los cuales están conformados por la siguiente información:

El Capítulo 1 tiene como propósito dar a la investigación un sistema coordinado y coherente de conceptos y proposiciones que permitan abordar el problema. Además de que se analiza las tendencias de las tecnologías y sistemas relacionados al tema investigado.

El Capítulo 2 muestra la solución propuesta para el problema planteado. Presentándose la descripción del sistema propuesto.

El Capítulo 3 representa la construcción de la solución propuesta donde se hace énfasis en la fase de análisis y diseño.

El Capítulo 4 se definirán los detalles más específicos de la implementación y se harán las pruebas para validar los resultados de la investigación realizada. Se modelará el diagrama de componentes para la implementación y se mostrarán los resultados de las pruebas realizadas.

Capítulo 1

Fundamentación Teórica

En este capítulo se reflejarán algunos conceptos importantes relacionados con estructuras de datos, así como las más utilizadas actualmente en el desarrollo de bibliotecas gráficas. Este estudio constituye el estado del arte existente actualmente de las estructuras de datos así como las aplicaciones que tienen las mismas en bibliotecas gráficas.

1.1. Estructura de datos

En el contexto de programación, una ED es una forma de organizar un conjunto de datos elementales (un dato elemental es la mínima información que se tiene en el sistema) con el objetivo de facilitar la manipulación de estos como un todo o individualmente de manera eficiente. Define la organización e interrelación de los datos y un conjunto de operaciones que se pueden realizar sobre ellos. Cada ED ofrece ventajas y desventajas en relación con la simplicidad y eficiencia para la realización de cada operación. De esta forma, la elección de la ED de datos apropiada para cada problema depende de factores como la frecuencia y el orden en que se realiza cada operación sobre los datos [Abreu, 2010].

Las ED se dividen en dos tipos, las lineales y no lineales, ver figura 1.1. Dentro de las lineales podemos encontrar dos grupos, las estructuras estáticas que engloba los arreglos, vectores y las dinámicas a la cual pertenecen las listas, mientras que las pilas y las colas varían en dependencia del tipo de implementación. Por otra parte están las estructuras no lineales que se dividen en dos grupos, las jerárquicas como son los árboles, y las no jerárquicas en la que se encuentran los grafos.

1.1.1. Lista enlazada

Una lista enlazada es una de las ED fundamentales, y puede ser usada para implementar otras estructuras de datos. Consiste en una secuencia de nodos, en los que se guardan campos de datos arbitrarios y una o dos referencias (punteros) al nodo anterior o posterior [Aaron M. Tenenbaum, 1993]. Existen dos tipos de listas: están las simplemente enlazadas y las doblemente enlazadas. En las primeras, cada nodo posee un campo que es de tipo puntero y que indica cuál es el próximo elemento de la lista, ver figura1.2. Las segundas poseen, además del campo que contiene la información del nodo, dos punteros: uno para indicar cuál es el próximo elemento de la lista y otro para saber cuál es el nodo anterior, ver figura 1.3. La diferencia es que la lista simplemente enlazada se puede recorrer sólo en un sentido, y la doblemente enlazada, en ambos sentidos [Bonanata, 2005].

El hecho de que todo nodo contiene al menos una referencia a una instancia de su mismo tipo, permite que varias instancias puedan ser conectadas y quedar dispuestas una a continuación de la otra formando una lista enlazada 1.2. Cada elemento de la lista se encapsula en un nodo, en su campo Información[**?**].

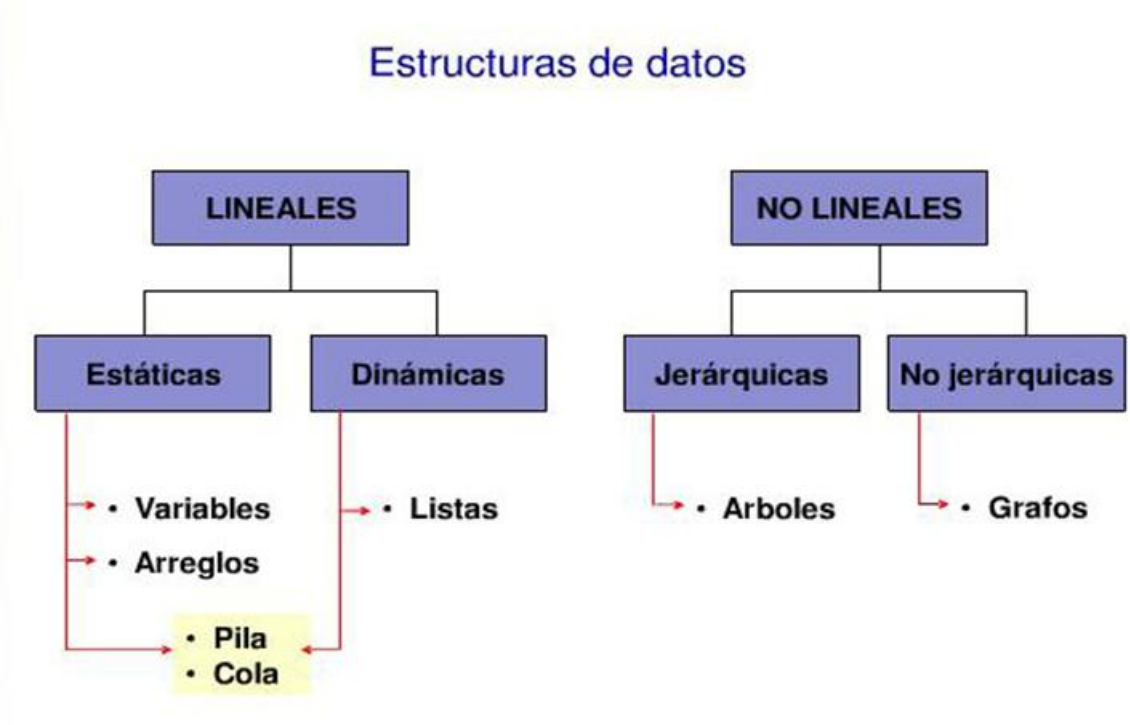

Figura 1.1: *Estructuras de datos*

Una de las ventajas de las listas doblemente enlazadas es que se pueden recorrer en ambos sentidos, o sea de un nodo podemos movernos al siguiente y al anterior, lo que puede aumentar la eficiencia de las operaciones.
Las listas enlazadas permiten ingresar elementos en cualquier posición sin necesidad de crear un espacio en memoria como en los vectores[Bonanata, 2005]. Permiten la inserción y remoción de los nodos en cualquier punto. Una desventaja es que una lista no es la estructura de datos adecuada para el acceso aleatorio. Cada vez que se busca un elemento o recorre la misma siempre se comienza desde el principio. Son usadas como módulos para otras muchas estructuras de datos, tales como pilas, colas y sus variaciones[Bonanata, 2005].

En muchos problemas se imponen restricciones a los elementos que se pueden almacenar en una lista, cuando una lista tiene todos sus elementos de un mismo tipo se dice que ella es **homogénea**, en caso contrario es **heterogénea**, lo que significa que en la lista puede haber elementos de diferentes tipos[**?**].

Esta es la descripción de las operaciones de la lista, la representación formal se muestra a continuación:

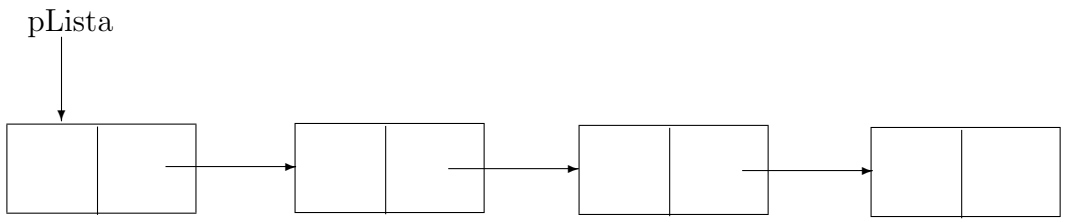

Figura 1.2: *Lista simplemente enlazada*

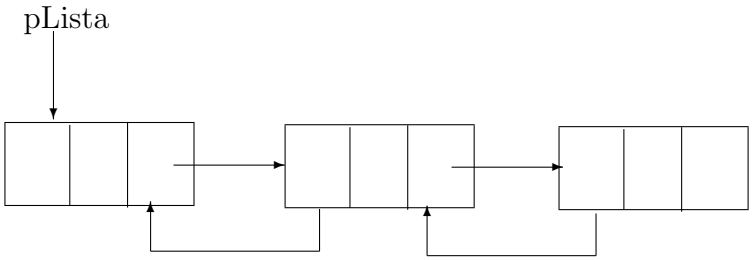

Figura 1.3: *Lista doblemente enlazada*

TDA **Lista**

Datos

Una colección ordenada de elementos del mismo tipo. Se le llama: l

Operaciones

Constructor

Crea una nueva lista, la crea vacía.

Insertar(**Tipo** x, **Entero** i)

Inserta un nuevo elemento x, en la posición i de la lista

Adicionar(**Tipo** x)

Adiciona un elemento al final de la lista.

Tipo Obtener(**Entero** i)

Devuelve el elemento que está en la posición i de la lista

Eliminar(Entero i)

Elimina el elemento que está en la posición i. Los elementos posteriores a partir de la posición i +1 pasan a tener la posición anterior inmediata es decir i

Entero Longitud( )

Devuelve la cantidad de elementos de la lista. Si la lista es vacía devuelve el valor cero.

Lógico Vacía()

Ejemplo:
L = (5, 4, 3, 2, 1)
L es **Homogénea**, todos los elementos son del mismo tipo
O = (5.10, 4.99, 3, 2, "1", (3,6,9))
O es **Heterogénea**, se mezclan elementos de diferentes Tipos, L1 es real y L5 es una cadena.

Figura 1.4: *Ejemplo de lista homogénea y heterogénea*

Devuelve un valor lógico verdadero si la lista tiene longitud cero, falso en caso contrario.
FIN

1.1.2. Pila

Una pila es una colección ordenada de elementos que puede insertarse o suprimirse por un extremo, llamado tope, nuevos elementos. Se puede representar una pila como en la figura 1.5. Una pila (del inglés *stack*) es una lista ordinal o estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (del inglés *Last In First Out*, último en entrar, primero en salir) que permite almacenar y recuperar datos. Se aplica en multitud de ocasiones en informática debido a su simplicidad y ordenación implícita en la propia estructura [Aaron M. Tenenbaum, 1993].

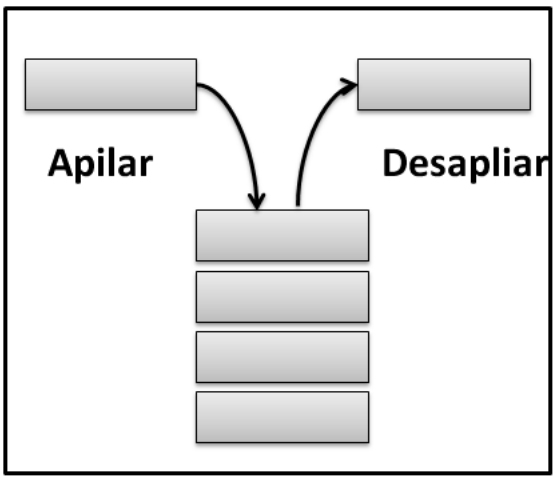

Figura 1.5: *TDA Pila*

La definición de pila incorpora la inserción y suspensión de elementos, de tal manera que ésta es un objeto dinámico constantemente variable. En consecuencia surge la pregunta: ¿Cómo cambia una pila? La definición que un extremo de la pila se designa como tope de la misma. Pueden agregarse nuevos elementos en el tope de la pila (en cuyo caso este se mueve hacia arriba para corresponder al nuevo elemento que ocupa la cima), o pueden quitarse los elementos que están en el tope y entonces el tope se mueve hacia abajo para corresponder al nuevo elemento que se encuentra hasta arriba tal como se muestra en la figura1.6 [Aaron M. Tenenbaum, 1993].

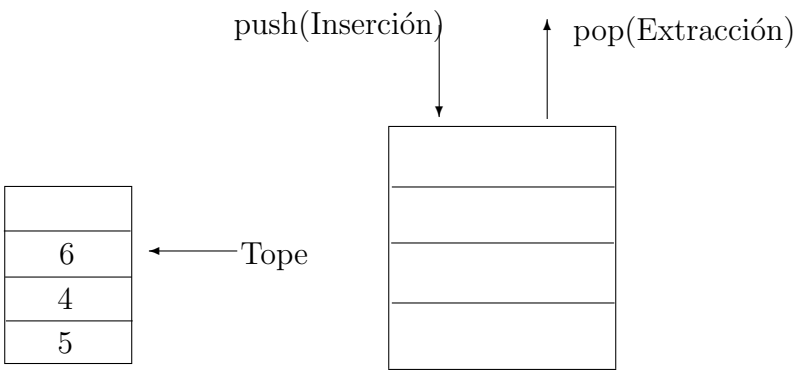

Figura 1.6: *Ejemplo de funcionamiento de una Pila*

Esta es la descripción de las operaciones de la pila, la representación formal se muestra a continuación:

TDA **Pila**

Datos

Secuencia ordenada de elementos, nombrada p

Operaciones

Constructor

Crea una pila vacía

Push(**Tipo** x)

Coloca a x en el tope de la pila. Garantiza que x = Top().

Tipo Top()

Devuelve el tope de la pila. Requiere que la pila no esté vacía.

Pop()
Elimina el primer elemento. Requiere que la pila no esté vacía.
Lógico Empty()
Devuelve verdadero si la pila está vacía.
FIN

La pila es una estructura muy utilizada en la solución de problemas. Una de sus aplicaciones más sencillas es invertir una cadena.

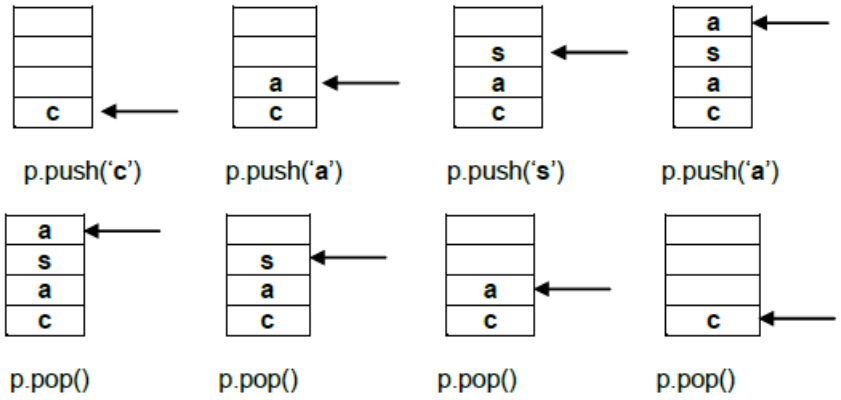

Figura 1.7: *Invirtiendo la cadena "casa" utilizando una pila*

Otra aplicación más compleja consiste en identificar si una expresión tiene sus paréntesis balanceados. Balanceados significa que para cada paréntesis abierto existe uno cerrado en el mismo orden.

(x+3)+(5*(4+2)) - ***PARENTESIS BALANCEADOS !!!!***
)(x+3)+(5*(4+2) - ***PARENTESIS NO BALANCEADOS !!!!***

1.1.3. Cola

Una cola es una ED, caracterizada por ser una secuencia de elementos en la que la operación de inserción push se realiza por un extremo y la operación de extracción pop por el otro [Aaron M. Tenenbaum, 1993], figura 1.8.

Las colas son EDs que permiten realizar dos operaciones básicas sobre ellas. Estas operaciones son agregar y quitar elementos de la misma. Las colas tienen la particularidad

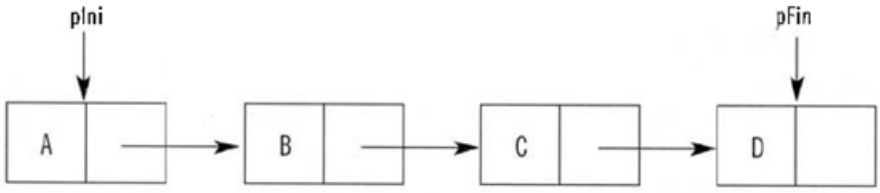

Figura 1.8: *Representación de una cola*

de que el orden en que se extraen los elementos es el mismo en el que entraron. Este comportamiento se llama FIFO (*First-In-First-Out*, el primero que entra es el primero que sale), ver la figura1.9 . Para llevar el control de la cola hacen falta dos punteros: uno que apunte al comienzo y otro que apunte al último nodo [Bonanata, 2005].

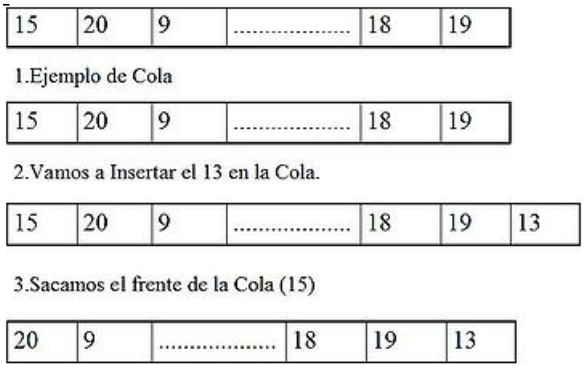

Figura 1.9: *Ejemplo de funcionamiento de una cola*

Las colas aparecen en diversos ámbitos de la actividad humana; la gente hace cola en los cines, en las panaderías, en las bibliotecas, etc. También tienen aplicación en la computación; una situación bien conocida es la asignación de una CPU a procesos de usuarios por el sistema operativo con el algoritmo para asignación de memoria *round-robin* sin prioridades, donde los procesos que pide el procesador se encolan de manera que, cuando el que se está ejecutando actualmente acaba, pasa a ejecutarse el que lleva más tiempo esperando [Gutiérrez, 1993].

El TDA **Cola**

Datos

Secuencia ordenada de elementos nombrada c

Operaciones

Adicionar(**Tipo** x)

coloca a x en la cola (después del ultimo elemento) de la cola.
Tipo Frente()
devuelve el elemento que se encuentra en la primera posición de la cola. Se dispara una excepción cuando la cola está vacía.
Tipo Fondo()
devuelve el elemento que se encuentra en la última posición de la cola. Se dispara una excepción cuando la cola está vacía.
Tipo Extraer()
elimina el elemento que se encuentra en la cabeza de la cola, si está vacía se dispara una excepción.
Lógico Vacía()
devuelve verdadero si la cola está vacía.
FIN

Cola con prioridad

Las colas con prioridad tienen la característica de estar basadas en la estructura de datos denominada montículo. La característica principal es que cada elemento introducido está asociado a una prioridad. A medida que se van introduciendo valores se van organizando para que al recuperar uno, siempre se recupere primero, el que tiene menos prioridad. La aplicación típica de ésta cola es por ejemplo, el organizar una lista de impresión por prioridad a medida que se introducen y procesan trabajos, siempre se procesa primero el de menor prioridad.

1.1.4. Árbol

En ciencias de la informática, un árbol es una ED ampliamente usada que imita la forma de un árbol (un conjunto de nodos conectados). Son estructuras de datos conformadas por nodos en forma jerárquica. Cada nodo, además de contener información, tiene una determinada cantidad de punteros que nos indican el grado del árbol. Se dice que los árboles se basan en relaciones padre-hijo, véase la figura 1.10. Estas relaciones se componen a partir de un nodo raíz, que es el único nodo que no posee un padre. Se dice que un nodo es padre de otro cuando uno de sus punteros apunta a otro llamado nodo hijo. Todo nodo, menos el raíz, tiene al menos un padre. Véase, en la Figura 1.8, un diagrama de un árbol binario con nodos dinámicos [Bonanata, 2005].

Los árboles se caracterizan por mantener una serie de datos ordenados en memoria, organizados de forma tal que tareas como inserción, búsqueda y eliminación de datos en

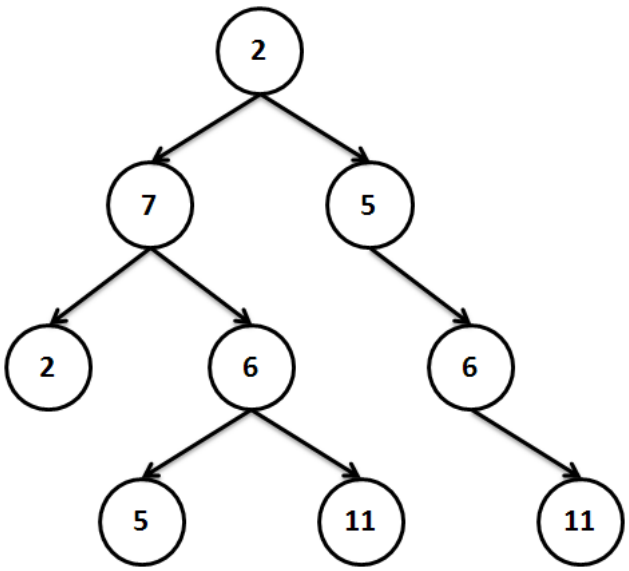

Figura 1.10: *Árbol*

él son, por lo general, más rápidas que en otras estructuras de datos, como pueden ser las listas, las pilas y las colas. El grado de un árbol indica cuántos hijos puede tener, como máximo, cada nodo. El árbol más conocido y utilizado es el árbol binario. Como su nombre lo indica, el grado de dicho árbol es dos. Esto significa que cada nodo en un árbol binario sólo puede tener dos, uno o ningún nodo hijo.

CrearNuevo()
Poscondición: Se crea un árbol binario vacío
EsVacio()
Poscondición: retorna verdadero si el árbol es vacío falso en caso contrario
Altura()
Poscondición: Retorna la altura del árbol
CopiaEn()
Poscondición: NuevoArbol tiene una copia del árbol
DevolverIzquierdo()
Precondición: El árbol binario no es vacío
Poscondición: Retorna una copia del subárbol izquierdo
DevolverDerecho()
Precondición: El árbol binario no es vacío
Poscondición: Retorna una copia del subárbol derecho
PreOrden()
Poscondición: Se obtiene la lista en preorden de los nodos del árbol
EntreOrden()
Poscondición: Se obtiene la lista en entreorden de los nodos del árbol
PostOrden()
Poscondición: Se obtiene la lista en postorden de los nodos del árbol
ALoAncho()
Poscondición: Se obtiene la lista de los nodos del árbol por niveles

---

Una aplicación de los árboles se encuentra en los juegos y un participante es la computadora. Donde el mismo es utilizado para calcular la mejor jugada que la computadora pueda hacer o escoger la mejor opción desplazarse por un tablero, un ejemplo de esto es el juego del gato y el ratón [Aaron M. Tenenbaum, 1993].

1.1.5. Grafo

Un grafo consta de un conjunto de nodos (o vértices) y un conjunto de arcos (o aristas). Cada arco de un grafo se especifica mediante un par de nodos. La figura 1.11 ilustra un grafo. Si el par de nodos que constituyen el arco son pares ordenados, se dice que el grafo es **n** grafo dirigido (o dígrafo).Obsérvese que un grafo no es por fuerza un árbol, pero un árbol es un grafo. Un aspecto a tener en cuenta es que un nodo no necesita tener ningún arco asociado al él [Aaron M. Tenenbaum, 1993].

Por analogía con los árboles, distinguimos diferentes recorridos en profundidad, un recorrido en anchura y el llamado recorrido por ordenación topológica. Todos ellos usan conjuntos para almacenar los vértices visitados en cada momento porque, a diferencia de los árboles, puede accederse a un mismo nodo siguiendo diferentes caminos y deben evitarse las visitas reiteradas [Gutiérrez, 1993].

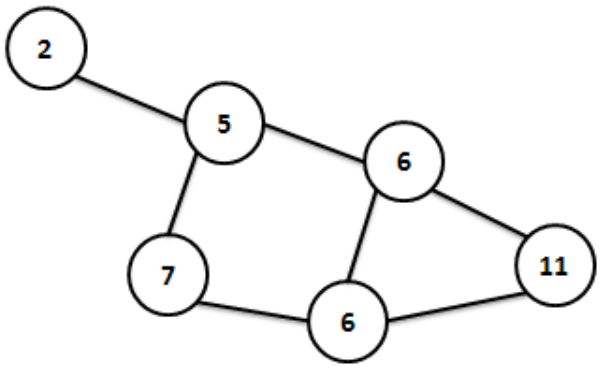

Figura 1.11: *Grafo*

A continuación se abordarán las principales responsabilidades y operaciones que se realizan sobre el TDA Grafo.

Crear()
Construye un grafo vacío, es decir, un grafo sin vértices ni aristas.
EsVacio()
Retorna verdadero si el grafo es vacío, retorna falso en caso contrario.
NumeroDeVertices()
Retorna el número de vértices del grafo.
NumeroDeAristas()

Retorna el número de aristas del grafo.

EstaElVertice (Vértice)

Retorna verdadero si existe un vértice en el grafo Igual a Vértice, es decir, si existe un vértice en el grafo cuya información es igual a la información almacenada en Vértice. Retorna falso en caso contrario.

EstaElArco (**Vertice1**, **Vertice2**)

Retorna verdadero si el grafo contiene un par de vertices V1 y V2 tal que: V1 es Igual a Vértice1 V2 es Igual a Vértice2 ¡V1, V2¿es un arco en caso contrario retorna falso.

InsertarVertice (Vértice)

Inserta en el grafo un nuevo vértice con la información contenida en Vertice.

InsertarArco (Vertice1, Vertice2)

Precondición: El grafo contiene un par de vértices tal que: V1 es Igual a Vértice1 V2 es Igual a Vértice2 ¡V1, V2¿no es un arco de G Poscondición: ¡V1, V2¿pasa a ser un arco del grafo.

Eliminar(Vértice)

Precondición: El grafo posee un vértice V que es Igual a Vértice Poscondición: V y todos los arcos en que él interviene son eliminados del grafo.

EliminarArco (Vertice1,Vertice2)

Precondición: El grafo contiene un par de vértices tal que: V1 es Igual a Vértice1 V2 es Igual a Vértice2 ¡V1, V2¿es un arco Poscondición: ¡V1, V2¿deja de ser un arco en el grafo

AdyacentesA (Vértice, ListaDeAyacentes)

Precondición: El grafo incluye un vértice V que es Igual a Vértice Poscondición: ListaDeAdyacentes es una lista que contiene a todos los vértices adyacentes a V.

Reemplazar (VerticeViejo,VerticeNuevo)

Precondición: El grafo contiene un vértice V1 que es Igual a VerticeViejo y ningún vértice V del grafo es Igual a VerticeNuevo.

Poscondición: V1 es sustituido por VerticeNuevo

EsConexo()

Retorna verdadero si el grafo es conexo, retorna falso en caso contrario.

RecorrerALoAncho

(ListaDeVertices)

Poscondición: ListaDeVertices es una lista que contiene todos los vértices del grafo en el orden en que fueron visitados según la estrategia de recorrido a lo ancho.

RecorrerEnProfundidad

(ListaDeVertices)

Poscondición: ListaDeVertices es una lista que contiene todos los vértices del grafo en el orden en que fueron visitados según la estrategia de recorrido en profundidad.

MenorCamino(Vertice1,Vertice2,ListaDelCamino)

Precondición: El grafo contiene un par de vértices tal que: V1 es Iguala Vértice1 V2 es Igual a Vértice2 V1 no es Igual a V2

Poscondición: ListaDelCamino es una lista que contiene el camino de menor longitud que va de V1 a V2. Si no hay camino ListaDelCamino es vacía.

1.1.6. Grafo de Escena

Un grafo de escena es un grafo dirigido acíclico de nodos que contiene los datos que definen un escenario virtual y controlan su proceso de dibujado. Contiene descripciones de bajo nivel de la geometría y la apariencia visual de los objetos, así como descripciones de alto nivel referentes a la organización espacial de la escena, datos específicos de la aplicación, transformaciones, etc. Los grafos de escena almacenan la información del escenario virtual en diferentes tipos de nodos. Existen nodos que almacenan la información geométrica y actúan como nodos hijos dentro del grafo de escena; el resto de los nodos suelen aplicar algún tipo de modificación sobre el segmento de jerarquía que depende de ellos, bien sea estableciendo agrupaciones, aplicando alguna transformación afín o realizando algún tipo de selección sobre alguna de sus ramas hijas. El proceso de dibujado consiste en realizar un recorrido de dicho grafo, aplicando las operaciones indicadas por cada tipo de nodo[Román, 2009].

Un grafo de escena no es un motor de simulación, aunque puede ser uno de los principales componentes de este motor, es objetivo principal es la representación eficiente de escenarios 3d[1]. La física, la detección de coliciones y el audio se deja a otras bibliotecas que utilice el programador[Martz, 2007].

El Grafo de Escena tiene como funciones principales:

- Contribuir a establecer una organización lógica de la escena.
- Establecer dependencias jerárquicas entre distintos sistemas de referencia.
- Posibilitar el proceso de selección entre múltiples niveles de detalle.
- Posibilitar el proceso automático de Culling (eliminación automática de los objetos que se encuentran fuera del campo de visión).

[1]Tres dimensiones

- Facilitar el control de la escena por parte del usuario.
- Hacer más cómodo el acceso a las bibliotecas gráficas de bajo nivel.

En la figura 1.12 se puede apreciar la descomposición de un objeto en sus diferentes componentes, de manera agrupada, lo cual sería una aproximación al grafo de escena que lo definiría.

Figura 1.12: *Grafo de escena*

En un grafo de escena pueden existir varios tipos de nodo, entre lo que se encuentran los siguientes:

- **Nodo de Geometría.** Almacenan la información poligonal de los objetos, también almacenan informaciones referentes a su apariencia, tales como material, textura, etc. Usualmente actúan como nodos hoja.[**?**]
- **Nodo Grupo.** Se emplean para agrupar varios nodos hijos, bien sea a nivel meramente organizativo, o para facilitar el proceso de culling jerárquico.[**?**]
- **Nodo Nivel de Detalle.** Usualmente llamados nodos LOD (Level of Detail). Seleccionan uno de sus hijos, basándose en la distancia entre el objeto con múltiples niveles de detalle y el punto de vista.[**?**]
- **Nodo de Transformación Afín.** Permite aplicar una matriz de transformación que afectara a ubicación espacial de sus nodos hijos. Son necesarios para la definición de objetos móviles y también para la creación de estructuras articuladas.[**?**]
- **Nodo de Switch.** Permiten realizar una selección entre sus nodos hijos.[**?**]

Debido a la organización jerárquica con que cuenta el grafo de escena, las funciones de acceso y de control son mucho más sencillas que en otras ED estudiadas. Esto se debe a que la posición de cada objeto en un escenario 3d sera relativa a la de sus antepasados. De esta forma la ubicación absoluta de un nodo en una escena dependerá de la ubicación del padre y así sucesivamente hasta la raíz[Junker, 2006].

1.2. Motores gráficos. Estructura de datos

Un motor gráfico no está completo sin algún modo de gestionar sus datos. Para algunos motores solo es necesaria la utilización de una simple lista para almacenar los elementos en escena, pero en la actualidad, en el mejor de los casos, dicha lista no es suficiente, por lo que se hace necesario el uso de una ED potente, lógica y óptima.

Mientras que algunos motores, por ejemplo G3D [Team, 2006], optan por no brindar ninguna ED o ninguna base común para dichas estructuras, otros plantan las bases para la implementación de estructuras por parte del usuario, como Ogre3D [Junker, 2006]. Algunos ofrecen además un grupo de nodos especializados como parte del motor, por ejemplo WildMagic [Everly, 2006]. Por otra parte otros optan por brindar la clase base para la construcción del grafo de escena, así como algunos nodos especializados para objetos genéricos (geometrías, luces y cámaras), dejando en manos de usuarios o de la comunidad, la implementación de otros nodos [Román, 2009], un ejemplo de este es la SceneToolKit desarrollada en la Universidad de las Ciencias Informáticas.

1.2.1. Ogre3D

Ogre3D es un motor gráfico 3D flexible y orientado en escenas, implementado en C++. Es multiplataforma y utiliza el API de OpenGL o Microsoft DirectX, es compatible con una gran cantidad de pluggins desarrollados, y se trata de un software libre [Mateo, 2010]. Este motor gráfico implementa un grafo de escena donde, los *SceneNodes* (nodo de escena) realizan un seguimiento de la ubicación y orientación de todos los objetos que se le atribuyen. Al crear una entidad, no es añadida a la escena hasta que no se le adjunta a un *SceneNode*. Su función es ser un contenedor de entidades además de otros objetos como cámaras o luces. Además, se pueden crear jerarquías de nodos, es decir, los *SceneNodes* pueden ser padres de otros*SceneNodes*[Junker, 2006].

1.2.2. Delta3D

Delta3D es un motor gráfico que puede ser usado en juegos, simulación u otras aplicaciones gráficas, que está implementado en C++. Tiene integrado como módulos OpenSceneGraph [Martz, 2007], Open Dynamic Engine (ODE), Character Animation Library (CAL3D) y la biblioteca de audio OpenAL. Se puede utilizar en sistemas operativos Windows y Linux, utiliza el API de OpenGL y se trata de un software libre[Mateo, 2010].

1.2.3. Unreal Engine

Unreal Engine proporciona una plataforma y herramientas para el desarrollo de videojuegos 3D de vanguardia, también utilizado para la realización de simulaciones militares. Fue creado por Epic Games para el juego Unreal (1998), está implementado en C++, implementa su propio grafo de escena y utiliza un tipo de script diseñado por la empresa creadora, llamado UnrealScript. Es multiplataforma. Utiliza la API OpenGL o el de Microsoft DirectX, y se trata de un software comercial[Mateo, 2010].

1.2.4. Wild Magic

Wild Magic se basa en un grafo en forma de árbol, ya que sólo permite la herencia simple, lo que significa que cada nodo interno o nodo hoja tiene un único nodo padre.

El grafo Wild Magic está compuesto únicamente de nodos que se derivan de una sola clase espacial abstracta que a su vez deriva de la clase base del objeto . Los nodos raíz e intermedios de grafo son implementados como objetos de la clase nodo y los nodos hoja como objetos derivados de la clase de Geometría, que es también una clase abstracta.

El nodo raíz es un punto de acceso único a toda la escena para ser representada. Las hojas de geometría derivada del grafo son los objetos renderizables y los nodos interiores contienen los datos que se propaga desde la raíz hasta las hojas.. Estos datos incluyen transformaciones y efectos de representación global asociada a cada nodo del subgrafo que afectan a la raíz en un nodo en particular [Everly, 2006].

1.2.5. SceneTollKit

Una adaptación interesante del grafo de escena en la SceneTollKit, es que los objetos están independientes y los nodos del grafo hacen referencias a ellos. Esto permite que varios objetos en la escena sean nodos que referencian a una malla cargada una única vez, lo que garantiza además la modularización de los datos y la definición de las responsabilidades

de las clases entidades y controladoras. También se hicieron algunas adaptaciones para minimizar el recorrido por el grafo en cada fotograma. Como resultado, el uso del grafo de escena ha permitido la aplicación tanto de técnicas de optimización (frustum culling, oclusión y niveles de detalles), como de otras soluciones de diversas índoles, basándose en su estructura [Román, 2009].

1.3. Consideraciones generales

1.3.1. Listas

Hay pocas ventajas en el uso de las listas enlazadas en los motores gráficos debido a que no permite el acceso aleatorio a sus elementos, hay mejores formas de implementar éstas estructuras, por ejemplo con árboles y grafos. Sin embargo, cuando una lista enlazada es dinámicamente creada fuera de un subconjunto propio de nodos semejante a un árbol, son usadas más eficientemente para recorrer ésta serie de datos [Pomier, 2006]. Las listas aparecen muchos en la solución computacional de muchos problemas, ya sea para almacenar objetos de determinada clase como para almacenar objetos de diferentes tipos.

1.3.2. Pila

La pila no es la ED adecuada para la colección de información en motores gráficos debido a que la forma de acceso a sus elementos se hace de una sola forma, por el Tope. Esto provoca que en tiempo de ejecución para acceder a un elemento haya que ir vaciando la pila hasta encontrarlo y luego volverla a llenar.
Una de las ventajas de la pila es que con ella se puede implementar la recuperación de datos, lo que quiere decir que podemos recuperar información que hayamos perdido apilando en esta estructura de datos los estados del sistema para posteriormente poder recuperarlos.

1.3.3. Cola

La cola es una ED que por sus características no es recomendable usarla en aplicaciones gráficas. Como todas las ED lineales no permite el acceso aleatorio a los elementos que almacena. Al igual que la pila hace más costoso el acceso a sus elementos en tiempo real debido a que en cada operación de búsqueda o selección se necesita ir vaciando la cola hasta encontrar el elemento y llenarla.

Las ED lineales no son las más adecuadas para la colección de elementos en un motor gráfico debido a que no establecen jerarquías entre objetos, lo que dificulta el manejo de escenas complejas tanto para el programador como para la computadora.

1.3.4. Árbol

El árbol es una ED con una amplia variedad de aplicaciones en la computación. La filosofía del árbol es bastante usada en aplicaciones de realidad virtual como son los juegos de tableros, softwares educativos, etc. Aunque es una estructura de datos jerárquica no establece dependencias entre distintos tipos de referencias. Esto significa que aunque se basa es la relación padre-hijo, estos últimos no heredan las propiedades de sus antepasados, en fin los padres solo contienen a los hijos así como los hijos contienen un puntero de de sus padres.

1.3.5. Grafo

Los motores gráficos implementan el esta ED para representaciones de entornos los definidos para una entidad en la escena los cuales determinan la posición de un elemento en una escena. Un ejemplo de esto son los GPS implementados para videojuegos, los cuales implementan un grafo de camino para seguir la ubicación de un auto o de un personaje en la escena[Abreu, 2010].

1.3.6. Grafo de escena

El grafo de escena es una ED ampliamente usada en la mayoría de los motores gráficos actuales. Lo anterior expresado y las características que presenta dicha ED hace que sea la más adecuada para das solución al problema planteado en el presente trabajo.

Capítulo 2

Solución Propuesta

Este capítulo tiene como objetivo fundamental mostrar la información con una visión más clara de la propuesta a defender dándole solución a la actual problemática en cuestión. Durante el desarrollo del mismo se justificarán las tecnologías, se expondrá una propuesta inicial de la solución del problema planteado, donde se explicará la elección de la estructura de datos que se escogió para el almacenamiento y manipulación de los elementos en la escena, así como una clara descripción de sus principales funcionalidades.

2.1. Descripción del problema

En el epígrafe se justifica la necesidad de implementación de una estructura de datos jerárquica para la biblioteca GLSVe, luego se explican las funcionalidades necesarias que se debe brindar.

2.1.1. Justificación de la implementación de una estructura de datos jerárquica para la GLSVe

La biblioteca GLSVe, para la gestión de los objetos en la escena no contaba con una ED óptima. Esto hizo que el acceso y la búsqueda a dichos objetos se hiciera más costoso tanto para el desarrollado como para la computadora. Por lo que se hizo necesario implementar una ED jerárquica que diera solución a los problemas planteados.

2.2. Graphics Library for Stereoscopic Vision (GLSVe)

GLSVe es una biblioteca estructurada en clases que permite representar objetos a distintas profundidades, que se observen por delante del monitor, en este, o detrás, según los paralajes correspondientes, brinda una interfaz fácil e intuitiva para su uso. La biblioteca ha sido desarrollada en el lenguaje C*sharp*, utilizando el recubrimiento Tao Framework para emplear las bibliotecas OpenGL la cual brinda facilidades para implementar las técnicas estereoscópicas con los elementos geométricos a tener en cuenta.

2.3. Selección de una Estructura de datos

Estructura de datos	Lineal	Jerárquica	Uso en Motores Gráficos
Listas Enlazadas	Si	No	No
Pilas	Si	No	No
Colas	Si	No	No
Árbol	No	Si	No
Grafo de Escena	No	Si	Si
Grafo	No	No	No

Tabla 2.1: *Tabla de características de Estructuras de datos*

En el capítulo 1 se realizó un estudio de las ED lineales y jerárquicas así como también se abordaron una serie de motores gráficos con el objetivo de caracterizarlos atendiendo a la estructura de datos que utilizan para la gestión de los objetos. Teniendo en cuenta las características de las estructuras de datos representadas en la tabla 2.1, finalmente se concluye que para el desarrollo del presente trabajo se implementará un grafo de escena, una estructura jerárquica que facilita el control de una escena por parte del usuario, contribuye a establecer la organización lógica de la escena, hace más cómodo el acceso a las bibliotecas gráficas de bajo nivel y es una estructura muy abarcadora empleada en la mayoría de programas gráficos 3d actuales.

2.4. Grafo de Escena

2.4.1. Estructura del Grafo de Escena

El Grafo de Escena se basa en un principio conocido como estado de herencia, donde los nodos internos representan el estado del sistema referente a una serie de propiedades que contiene cada nodo en la escena además de contener una colección de nodos que pueden ser nodos de grupo o de geometría. El estado de herencia es una característica del grafo de escena que dicta que cada nodo hereda las propiedades y transformaciones del estado de todos sus antepasados, en una línea directa con el nodo raíz. Por otra parte los nodos externos o nodos hojas representarán, mediante los modelos (*Model*) que implementa la GLSVe, la geometría del sistema. En cada nodo (*Node*) del grafo de escena se guarda la información espacial del objeto de nombre, posición, orientación, escala, visibilidad y estado geométrico. Los nodos se pueden agrupar en un *GroupNode*, que almacena la lista de sus hijos así como una serie de transformaciones que pueden ser hechas en la escena. Esto indica que, los nodos padre (que en este caso son los *GroupNodes*) contendrán espacialmente a sus nodos hijos, tal como muestra la figura 2.1.

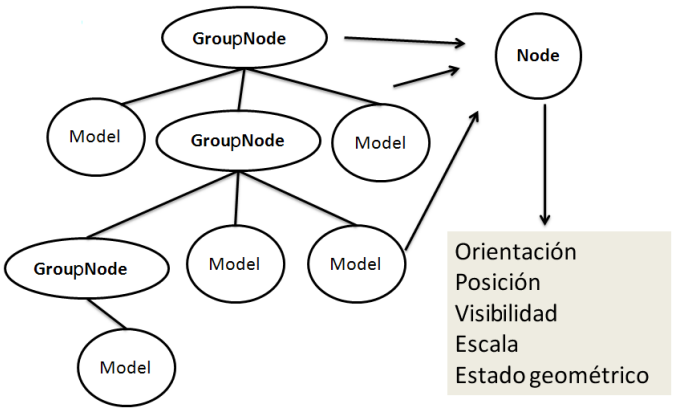

Figura 2.1: *Grafo de GLSVe*

2.4.2. Tipos de Nodos

El grafo de escena contiene diferentes tipos de nodos. La escena por lo general se compone de variedades de información. Los nodos deben contener todos los datos. Estos son los tipos de nodos estándar con los que cuenta la GLSVe(figura2.1):

- .Nodo de geometría: contiene los datos de los vértices, superficie normal o la descripción de la forma de los objetos.
- Nodo de dimensión: tiene la posición del objeto, la transformación, el tamaño y los datos de orientación.
- Nodos de grupo: determinará la organización jerárquica y composición de los otros nodos.

Nodo	Tipo de Nodo	Descripción
Node	Nodo de dimensión	*Clase padre de todos los nodos. Contiene las descripciones de alto nivel de todos los nodos.*
GroupNode	Nodo de Grupo	*En el grafo de escena solamente las hojas contienen objetos, el resto de los nodos intermedios constituyen agrupaciones de otros nodos representados a través de la clase* ***GroupNode***
Model	Nodo de Geometría	*Representa las geometrías en escena (puede contener una malla, una polilínea, una esfera, etc.).*

Tabla 2.2: *Tabla de tipos de nodos y sus descripciones*

2.4.3. Operaciones que brinda el Grafo de Escena

Al implementarse un grafo de escena fue necesario definir una serie de operaciones para facilitar el manejo de los elementos en la escena aprovechando las características de dicho grafo.

- **Adición de nodos**
  La adición de nuevos nodos puede verse desde dos puntos de vista, el primero es dándole a un nodo el padre en que será adicionado(con la función **IAtachTo**) de modo que si creamos un nuevo nodo **B** y le decimos que su padre será el nodo **A**, el nodo **B** se agregará a la lista de hijos de este último. Por otra parte con la función **IAtachNode** podremos adicionarle nuevos hijos a cualquier nodo en la escena En cualquiera de los casos el proceso de adición en el grafo se hace en dos pasos fundamentales:
  1. Primeramente se chequerá la existencia del nodo padre o raíz donde se insertará el nuevo nodo o hijo.
  2. Luego se chequerá la existencia del nodo en el grafo, de no estar, pasa a ser insertado, de lo contrario se lanza una excepción avisando de la existencia del nodo en el grafo.
  Esto asegura de que no existan nodos repetidos en la escena.

- **Sustracción de nodos**
  El proceso de sustracción de los nodos parte de que un nodo determinado le pide al padre, con la función **DetachFromParent**, que lo elimine de la lista de sus hijos. Este último buscará el nodo dentro de sus hijos y con la función **PkDetachNode** lo eliminará de dicha lista. Hay que destacar que por el estado de herencia que implementa el grafo de escena, al ser eliminado el nodo todos sus hijos también serán eliminados. La figura 2.2 muestra el proceso de sustracción de un nodo del grafo

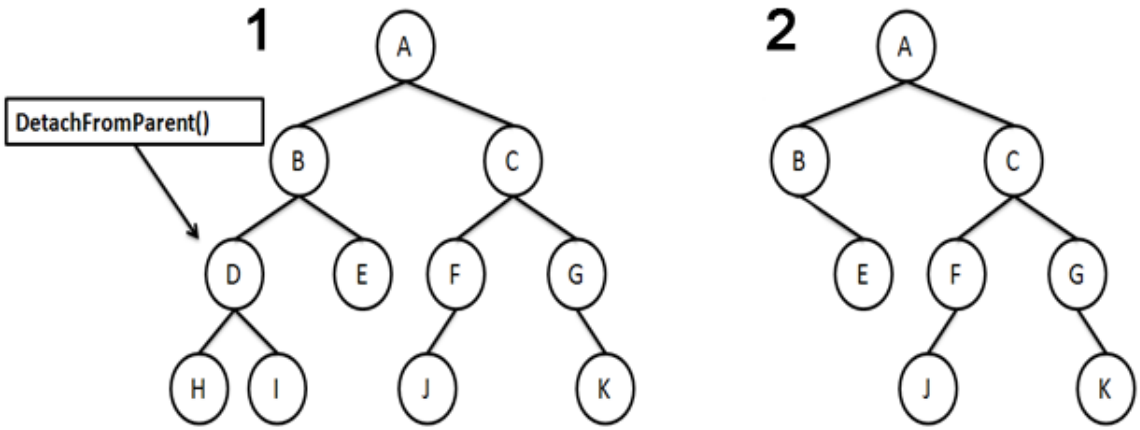

Figura 2.2: *Estado del grafo de escena luego de eliminar un nodo*

- **Búsqueda en el Grafo**
  Existen varias formas de búsqueda de el grafo. Está la búsqueda de un hijo de un nodo determinado (**IFindChildNode**) al cual se puede acceder mediante los atributo nombre (*NodeName*) o un identificador (*NodeID*), este tipo de búsqueda devuelve la posición en que se encuentra el nodo en la lista de hijos(*mApkChildrenList*) del nodo padre. Por otra parte, la búsqueda(**PkFindNode**) es mas general, sin embargo, es similar a la anterior, pero esta vez se realiza un recorrido del grafo entero. Dicha búsqueda siempre parte desde el nodo raíz haciendo un recorrido en preorden, visitando todo el grafo hasta encontrar en nodo buscado. Si al final de dicha búsqueda hubo resultados, se retornará un puntero del nodo, de lo contrario el valor devuelto será nulo.

- **Gestión del dibujado**

  Para dibujar todos los objetos visibles de la escena en cada ciclo se recorre el grafo de escena según los pasos siguientes:

  1. Primeramente se recorre el grafo actualizando el estado espacial (posición, orientación y escala) de todos los modelos de la escena para su estado geométrico mundial y local(respecto a su padre). Existe un atributo *Active* que define si el nodo puede cambiar su posición o no en el tiempo. Si el nodo no está *Active* no se actualizan sus hijos, en este caso no se recorre esa rama.

  2. En un segundo paso se recorre el grafo para dibujar los objetos de la escena según el modo estereoscópico activo:
  2.1 Si el modo estereoscópico (*Mode*) activo es **MonoMode** la escena se dibujará desde una sola cámara (*SingleEye*). Se procede a crear el frustum de la cámara y se define la mira (**lookat**) para este único ojo del observador. Se invoca el método **DrawHead**.
  2.2 Si el *Mode* activo es **StereoMode** la escena se dibujará desde las dos cámaras(*LeftEye* y *RightEye*). Se calcula el *frustum* y la mira (**lookat**) para ambos ojos del observador. Se activa el filtro asociado al modo estereoscópico activo y se invocan los métodos **DrawLeft** y **DrawRight** en el momento determinado, en función de la técnica activa.

- **Cambio de Padre de un Nodo**
  Esta función brinda la posibilidad de cambiarle el padre de un nodo determinado. Aprovechando las características de herencia con que cuenta el grafo de escena, un nodo cualquiera, excepto el nodo raíz(*SceneghraphRot*), puede cambiar de padre heredando todas las propiedades como orientación relativa, transformación, etc. De esta forma, si un nodo llamado nodo **C**, que tiene la posición (3,0,0), y su padre, llamado nodo **B**, que es el nodo raíz y, que por lo tanto, tiene sus coordenadas en escala absoluta, tiene la posición (2,0,0), la ubicación absoluta de **C** será la (5,0,0). Luego, si le cambiamos el padre al nodo **C**, por un nodo **A**, que tiene la posición (8,1,3,), la nueva ubicación absoluta de **C** será la (11,1,3), ver figura 2.3.

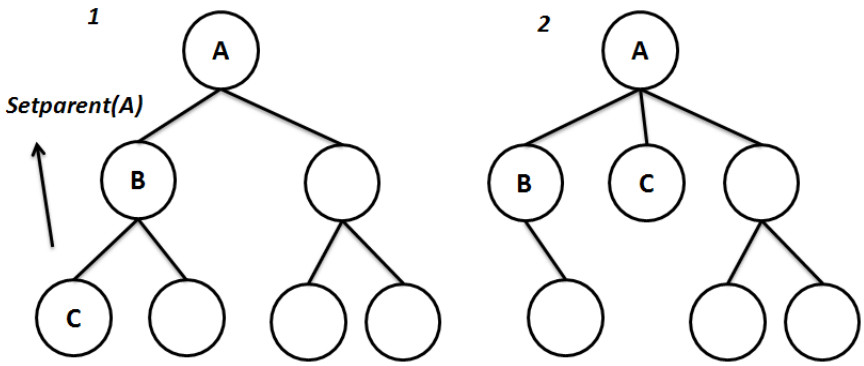

Figura 2.3: *Ejemplo del cambio de padre de un nodo*

- **Funciones de Orientación**

  . Función de Traslación

  La traslación de un objeto en la escena se realiza con la función **Traslate**, véase la figura 2.4, la cual actualiza un el vector de posición del nodo (*mKLocalPosition*) por un nuevo vector pasado por parámetro.

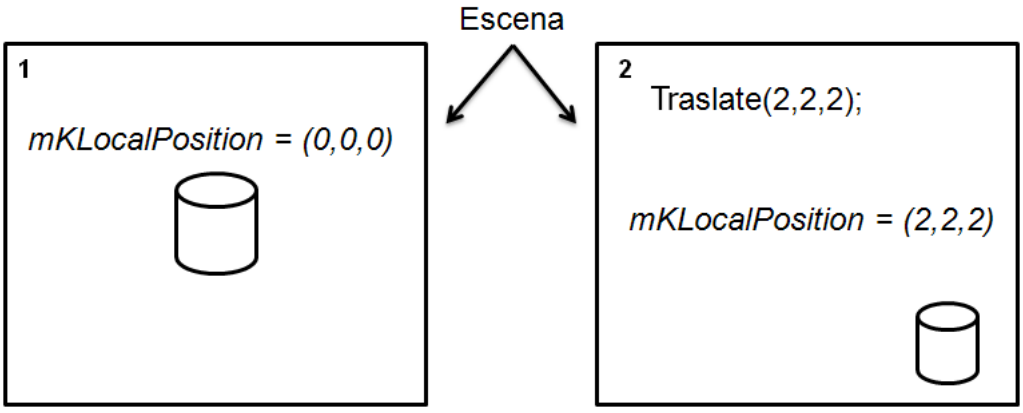

Figura 2.4: *Estado de un nodo al ser aplicado el Traslate*

. Función de Rotación

Por otra parte la rotación **Rotate** tiene un procedimiento similar al de traslación **Traslate**, pero esta vez se modifica la matriz de orientación(*mKLocalOrientation*) que posee el nodo.

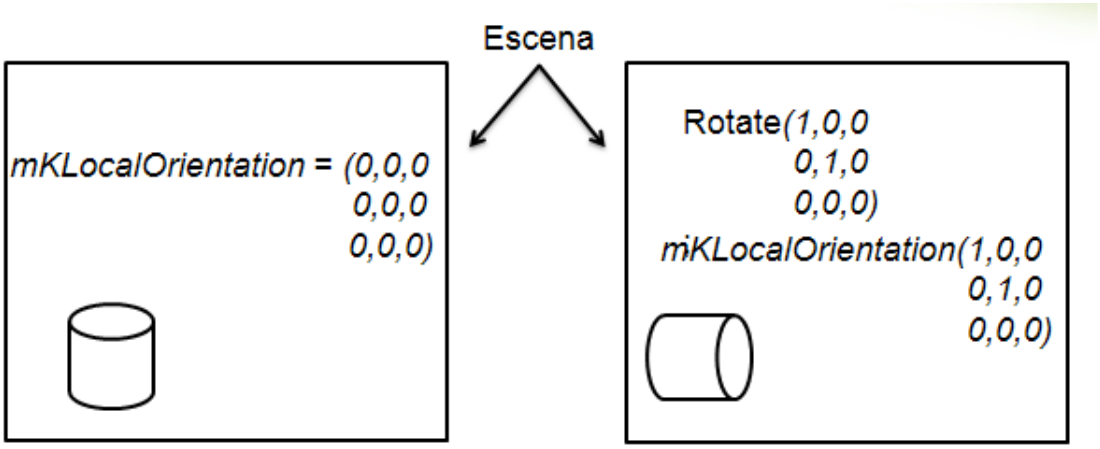

Figura 2.5: *Estado de un nodo al ser aplicado el Rotate*

. Función de rotación hacia un punto

Esta función hace rotar a un nodo hacia un vector posición (*argKTargetLocation*) dado.

Figura 2.6: *Estado de un nodo al aplicarle un LookAt*

. Función actualización del estado de las geometrías en la escena

Cada nodo del grafo tiene un atributo que dicta si puede ser actualizado o no (*mStatic*). Si el nodo no es estático se actualiza el estado geométrico mundial como orientación(*mKWorldOrientation*), posición (*mKWorldPosition*)y escala (*mKWorldScale*) con la función **UpdateWorldGs()** afectando tanto al nodo como a sus hijos con la función **UpdateChildrenGs()**. De ser un nodo estático pues no se actualiza el estado del nodo y por herencia tampoco el estado geométrico de sus hijos, véase la figura 2.7.

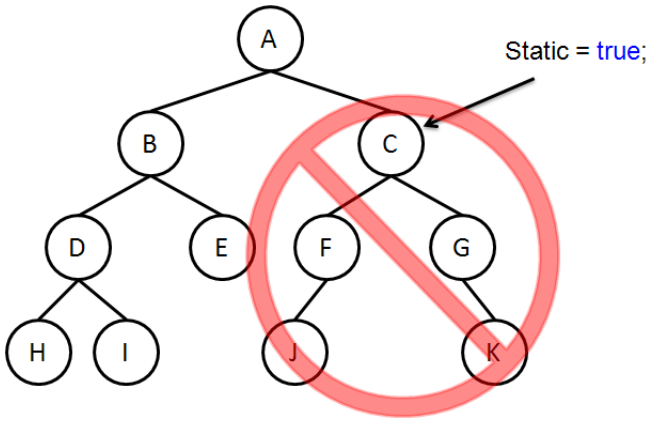

Figura 2.7: *Estado del grafo luego de ser actualizado*

2.4.4. Incorporación del grafo de escena a la GLSVe

La GLSVe es la encargada de seguirle la pista a todos los objetos que se dibujan por pantalla. Puede verse también como la encargada de todo lo que tenga que ver con el dibujo de la escena. La GLSVe posee el nodo raíz (*SceneGraphRoot*) de la jerarquía de nodos, ver figura 2.8, de forma que, a partir de él , se puede llegar a cualquier nodo de la escena. Esto significa que el primer nodo que se agrega al grafo de escena será un hijo de la raíz. Todas las operaciones que serán realizadas sobre el grafo de escena se harán desde la clase principal. La GLSVe actuará como mediador entre el programador y el grafo de escena.

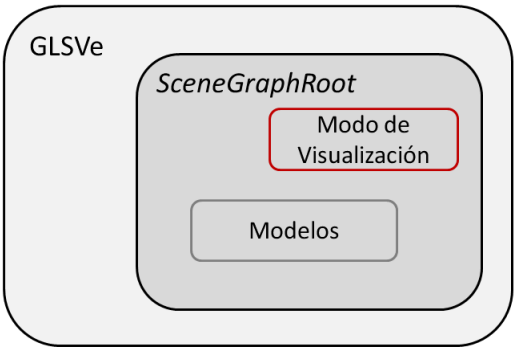

Figura 2.8: *Grafo de escena incorporado a la GLSVe*

2.5. Consideraciones generales

Durante el desarrollo de este capítulo se hizo una propuesta de solución al problema científico planteado. Después de haber hecho un análisis en el Capítulo 1 de las ED, se establecieron las bases para proponer una solución basada en un grafo de escena, ED que dará respuesta a la problemática planteada en la presente tesis.

Capítulo 3

Diseño de la solución

En el presente capítulo se tendrá como tema fundamental el análisis y diseño de la aplicación. Primeramente se definirá el modelo de dominio a utilizar. Luego se hará la captura de requisitos, que será utilizada posteriormente en la elaboración del modelo de casos de uso. Se definirán los actores del sistema y se especificará cada caso de uso obtenido a partir de la captura de requisitos. Se modelarán diferentes diagramas de secuencia del diseño con el objetivo de proporcionar una comprensión más profunda del sistema que se desea desarrollar.

3.1. Modelo de Dominio

El Modelo de Dominio es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. En el presente Modelo de Dominio la **GLSV** tiene un grafo de escena el cual puede estar vacío o almacenar una serie de nodos, dichos nodos pueden ser modelos del sistema(*Model*) o nodos de grupo (*GroupNode*). En la figura 3.1 se muestra el Modelo de Dominio elaborado.

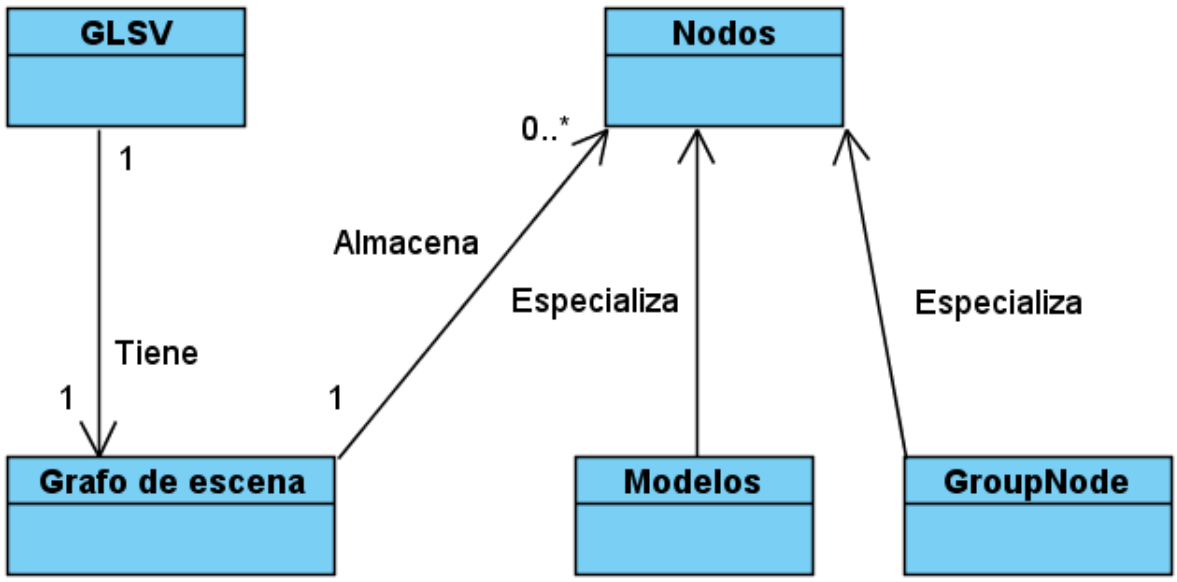

Figura 3.1: *Modelo de dominio*

GLSV es una biblioteca estructurada en clases para la visualización de escenas estereoscópicas.

Grafo de escena es la representación uno o un conjunto de nodos que comparten un espacio en la escena. Los mismos pueden ser gestionados, modificados o dibujados por la GLSV.

Node es una representación en el espacio de una serie de propiedades de un objeto como posición, orientación y escala.

GroupNode es la representación uno o un conjunto de nodos que comparten un espacio en la escena así como las propiedades de sus nodos antepasados.

Model es la representación geométrica de un objeto en un escenario ya sea una esfera, un cubo, una textura, un modelo 3D, etc.

3.2. Captura de Requisitos

Los siguientes epígrafes tienen como objetivo identificar los requisitos funcionales y no funcionales utilizados en la solución propuesta.

3.2.1. Requisitos Funcionales

Los requisitos funcionales representan la funcionalidad del sistema y se modelan mediante diagramas de casos de uso. Los siguientes requisitos responden a las funcionalidades que el sistema debe tener una vez concluida la implementación.

- **RF1.** Gestionar Nodos.
  - **RF1.1.** Adicionar Nodo.
  - **RF1.2.** Eliminar Nodo.
- **RF2.** Modificar Nodos.
  - **RF2.1.** Modificar Posición del Nodo.
  - **RF2.2.** Modificar Orientación del Nodo.
  - **RF2.3.** Modificar el Tamaño del Nodo.
  - **RF2.4.** Modificar el Padre del Nodo.
  - **RF2.5.** Modificar el Estado del Nodo.
  - **RF2.6.** Modificar la Visibilidad del Nodo.
- **RF3.** Buscar Nodo.
- **RF4.** Dibujar Nodo.

3.2.2. Requisitos no Funcionales

- **Software**
  El sistema operativo sobre el cual debe ejecutarse la aplicación será Windows XP y Windows 7.

- **Hardware**
  El modelo de microprocesador será Intel(R) Pentium IV a 3.0 GHz o superior. La memoria RAM será de 1GB.

- **Restricciones de Diseño e Implementación**
  Se empleará como lenguaje de programación C♯ y el framework Tao para la implementación y dibujado de los modelos.

3.3. Modelo de Casos de Uso 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 les pueda brindar a sus actores, es decir, los casos de uso del sistema.

3.3.1. Actores de Sistema

Actores	Justificación
GLSV	Es quien interactúa con el sistema para ejecutar las funcionalidades de: Gestionar los nodos de la escena, modificar sus propiedades así como buscarlos o dibujarlos.

Tabla 3.1: *Actor del sistema*

3.3.2. Diagrama de Caso de Uso del Sistema

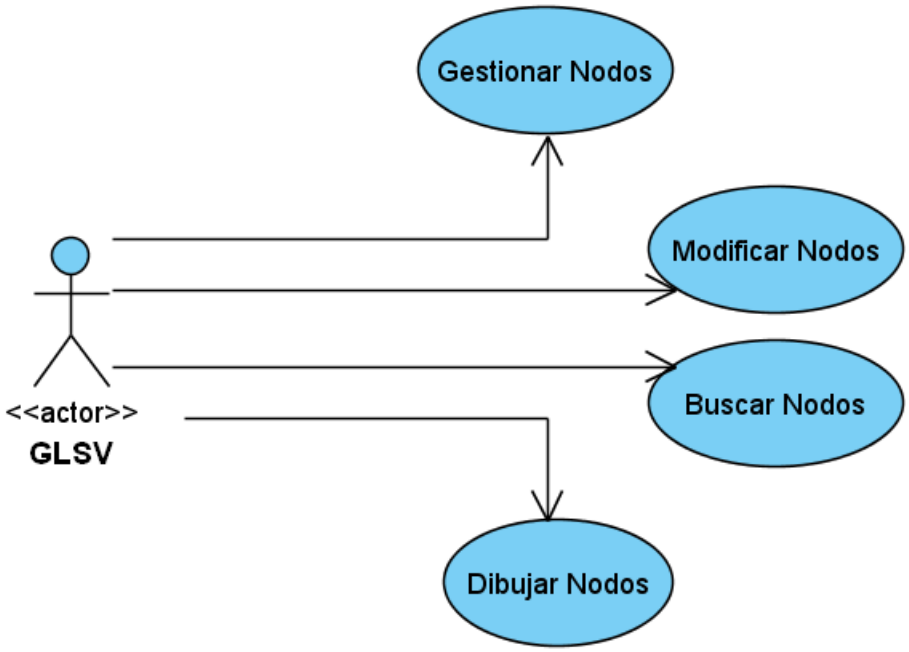

Figura 3.2: *Diagrama de Caso de Uso del Sistema*

3.3.3. Descripción de Casos de Uso del Sistema

Caso de Uso:	Gestionar Nodos	
Actores:	GLSV	
Propósito:	Su propósito es adicionar o eliminar los nodos en la escena.	
Resumen:	El caso de uso se inicia cuando la GLSV ejecuta una de las funcionalidades del grafo de escena	
Referencia:	RF1.1, RF1.2	
Flujo Normal de Eventos		
Acción del Actor		**Respuesta del Sistema**
1. La GLSVejecuta una de las funcionalidades de adicionar, eliminar, buscar o dibujar		1.1 El sistema ejecuta una de las siguientes funcionalidades: a) Si el actor desea adicionar un nodo consultar sección Adicionar Nodo. b) Si el actor desea eliminar un nodo consultar la sección Eliminar Nodo.
Postcondiciones:		Se adiciona un nuevo nodo en la escena o se elimina un nodo existente en la escena.
Prioridad:		Crítica.

Tabla 3.2: *Descripción del CU Gestionar Nodos*

Sección:	Adicionar Nodo
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. La GLSV ejecuta la funcionalidad adicionar.	2. El sistema verifica la existencia del nodo a adicionar.
Postcondiciones:	Se adiciona un nuevonodo en la escena.
Prioridad:	Crítica.
Flujos Alternos	
2.1 Si el nodo existe el sistema lanza un mensaje de error. 2.2 Si el nodo no existe el sistema adiciona el nuevo nodo.	

Tabla 3.3: *Descripción de la sección Adicionar Nodo del CU Gestionar Nodo.*

Sección:	Eliminar Nodo
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. La GLSV selecciona el nodo a eliminar y ejecuta la funcionalidad.	2. El sistema busca el nodo a eliminar 3. El sistema elimina el nodo.
Postcondiciones:	Se elimina un nodo de la escena.
Prioridad:	Crítica.

Tabla 3.4: *Descripción de la sección Eliminar Nodo del CU Gestionar Nodo.*

!

Caso de Uso:	Buscar Nodos	
Actores:	GLSV	
Propósito:	Su propósito es buscar los nodos en la escena.	
Resumen:	El caso de uso se inicia cuando la GLV ejecuta la funcionalidad del grafo de escena.	
Referencia:		
Flujo Normal de Eventos		
Acción del Actor		**Respuesta del Sistema**
1.La GLSVselecciona nodo que desea buscar y ejecuta la funcionalidad.		2. El sistema busca el nodo en el grafo de escena.
Postcondiciones:		Se busca un nodo en la escena
Prioridad:		Crítica.
Flujos Alternos		
2.1 Si el nodo existe el sistema devuelve un puntero con el nodo. 2.2 Si el nodo no existe el sistema lanza un mensaje de error.		

Tabla 3.5: *Descripción del CU Buscar Nodo.*

!

Caso de Uso:	Dibujar Nodos
Actores:	GLSV
Propósito:	Su propósito es Dibujar los nodos en la escena.
Resumen:	El caso de uso se inicia cuando la GLSV selecciona el modo de visualización y ejecuta la funcionalidad
Referencia:	
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. La GLSV selecciona el modo de visualización del grafo de escena 2. La GLSV ejecuta la funcionalidad.	3. El sistema actualiza el estado global y local de los nodos del grafo 4. El sistema dibuja los nodos del grafo de escena.
Postcondiciones:	Se dibuja la escena.
Prioridad:	Crítica.
Flujos Alternos	
3.1 Si el nodo esta visible el sistema lo dibuja en la escena 3.2 Si el nodo no está visible el sistema no lo dibuja.	

Tabla 3.6: *Descripción del CU Dibujar Nodo.*

!

Caso de Uso:	Modificar Nodos
Actores:	GLSV
Propósito:	Su propósito es modificar las propiedades de un nodo existente en la escena.
Resumen:	El caso de uso se inicia cuando la GLSV ejecuta una funcionalidad de modificación.
Referencia:	RF2.1, RF2.2, RF2.3, RF2.4, RF2.5, RF2.6
Precondición:	Debe haberse realizado el Caso de Uso Gestionar Nodos
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. La GLSV selecciona el nodo que desea modificar	2.El sistema busca el nodo que se desea modificar.
3. La GLSV ejecuta una de las funcionalidades para modificar la posición, la orientación, la escala, el estado, el padre o la visibilidad del nodo.	4. El sistema actualiza las propiedades del nodo seleccionado por la GLSV.
Postcondiciones:	Se modifican las propiedades de unanodo existente en la escena y se visualiza el resultado en la pantalla.
Prioridad:	Crítica.

Tabla 3.7: *Descripción del CU Modificar Nodos.*

3.4. Diagrama de Clases de Diseño

Figura 3.3: *Diagrama de Clases de Diseño*

3.5. Diagrama de Secuencia del Diseño

A continuación se representarán los diagramas de secuencia del diseño para tener una idea más general sobre el flujo que se realiza entre las clases del diseño y que posibilite comprender lo desarrollado en términos de implementación.

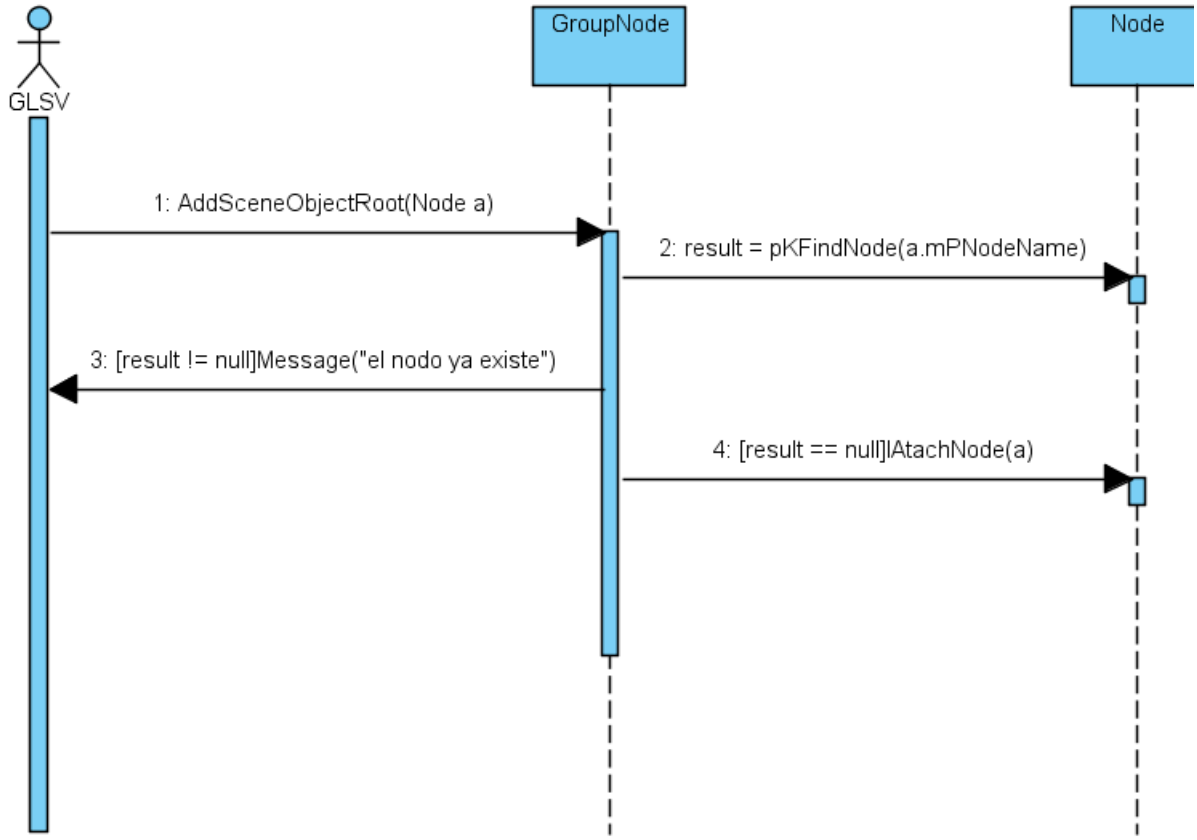

Figura 3.4: *Diagrama de Secuencia del CU Adicionar*

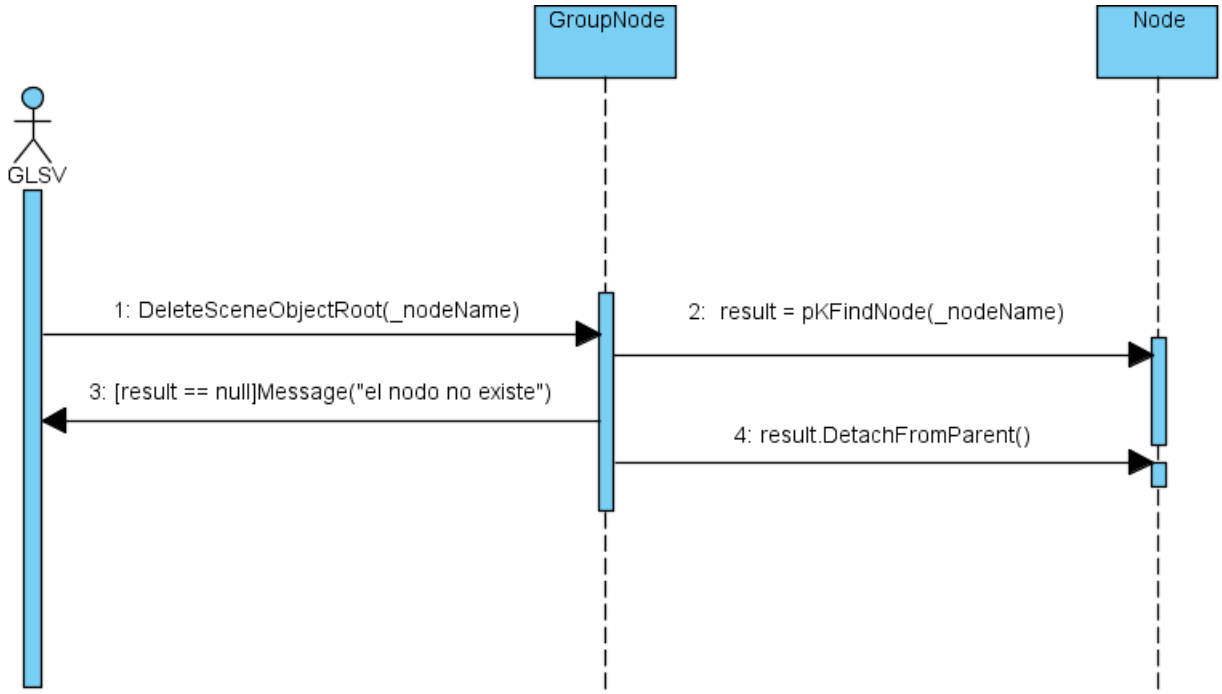

Figura 3.5: *Diagrama de Secuencia del CU Eliminar*

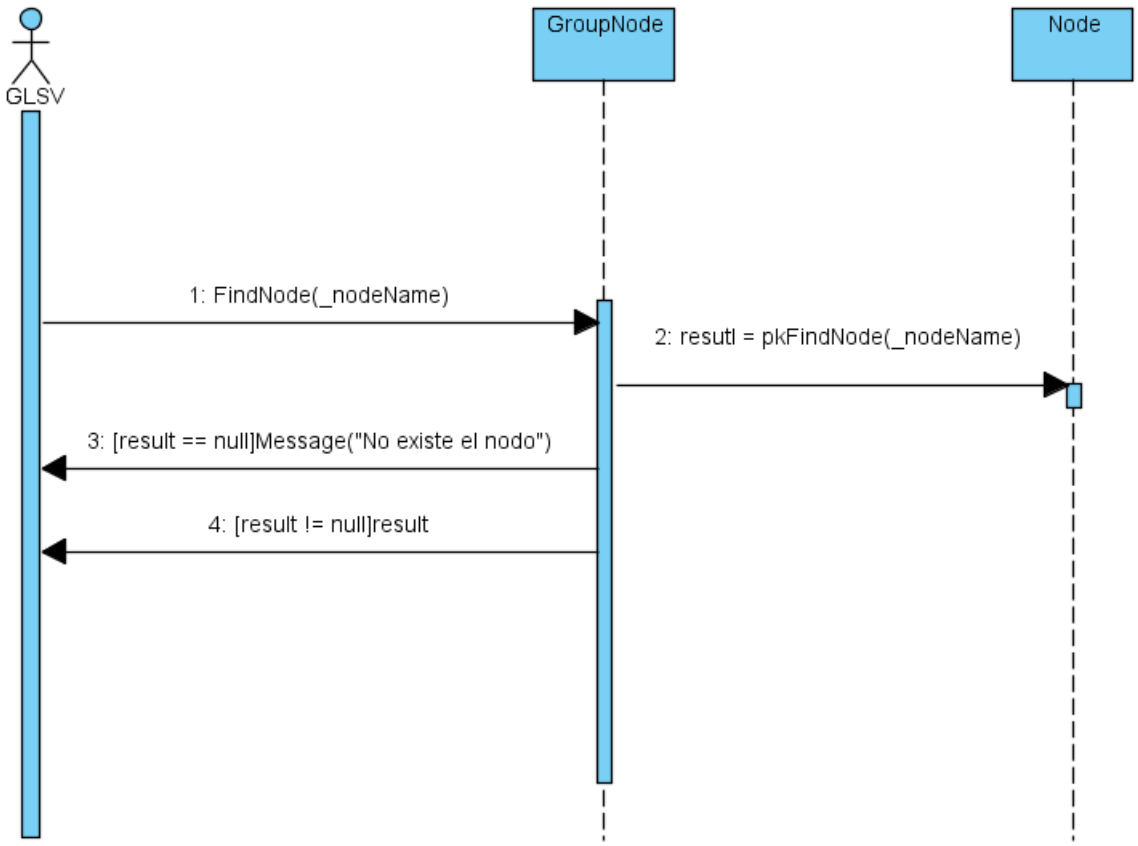

Figura 3.6: *Diagrama de Secuencia del CU Buscar*

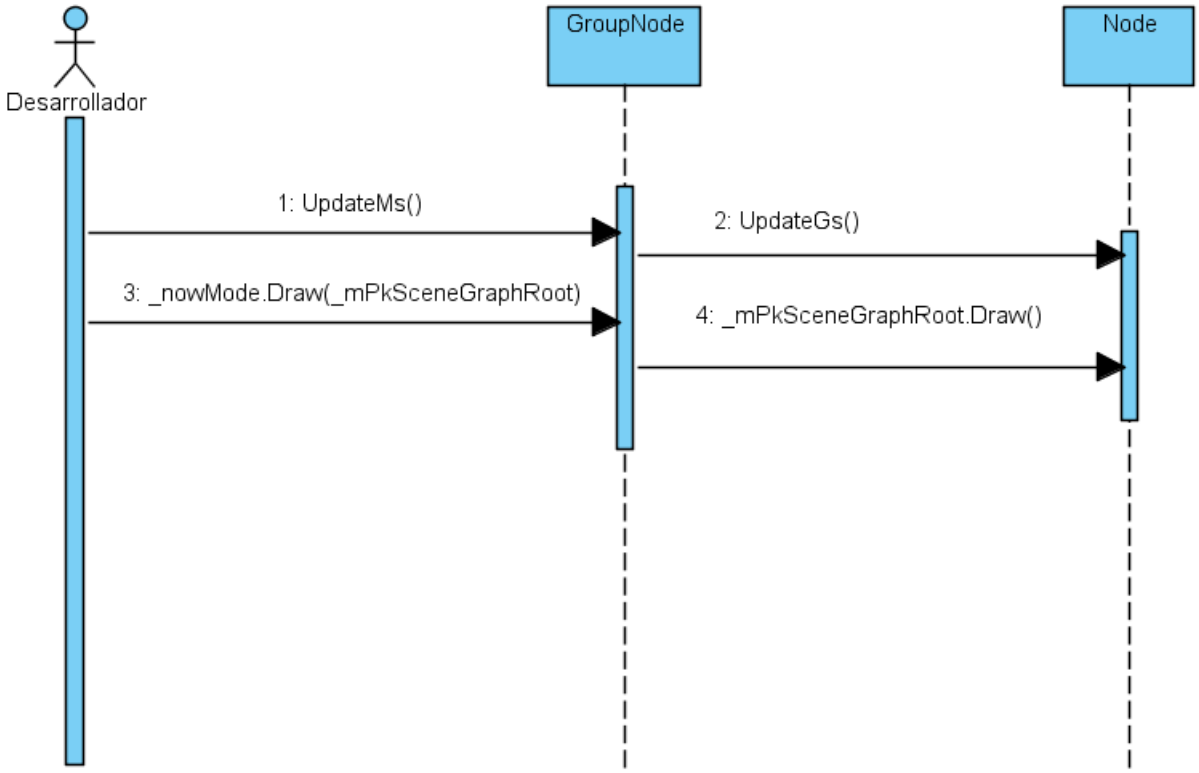

Figura 3.7: *Diagrama de Secuencia del CU Dibujar*

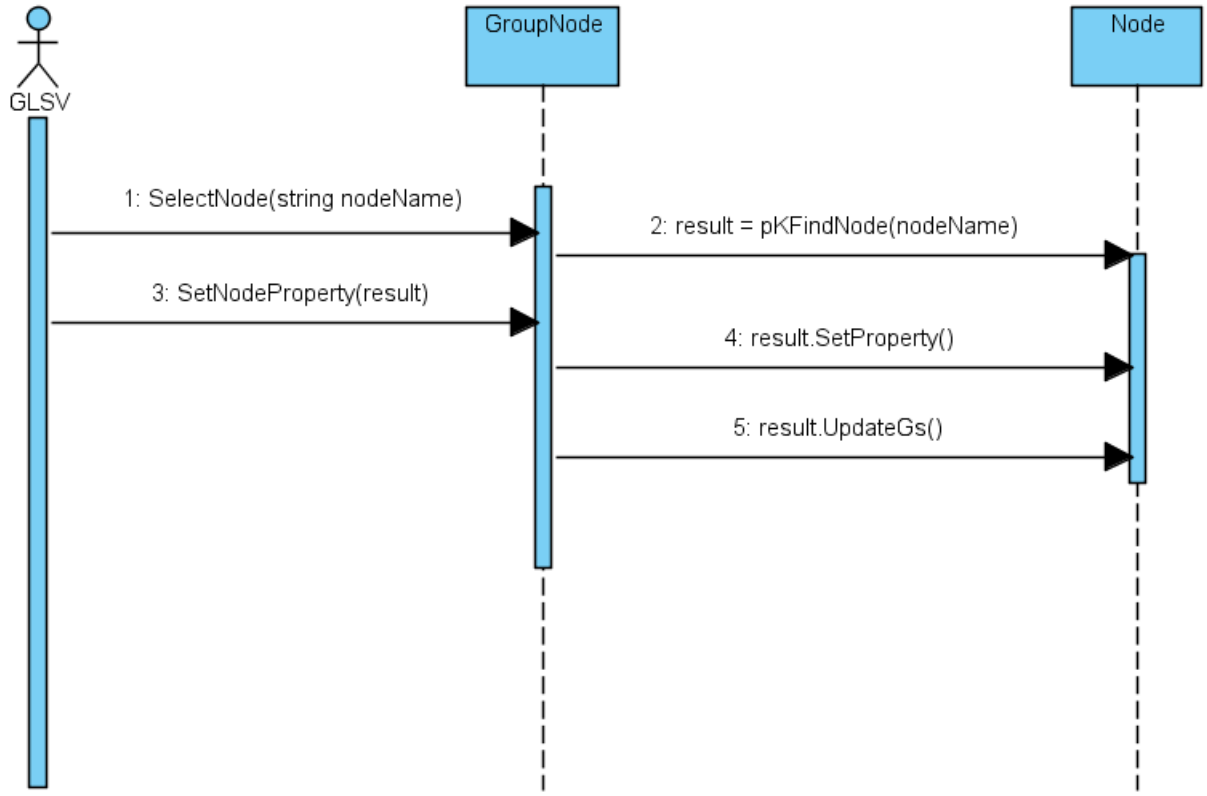

Figura 3.8: *Diagrama de Secuencia del CU Modificar*

Capítulo 4

Implementación y Análisis de resultados

En este capítulo se abordarán los temas de implementación del grafo de escena utilizando el trabajo realizado en los capítulos anteriores. Se modelará el diagrama de componentes utilizado en el grafo y se harán algunas pruebas para validar los resultados y medir aspectos como el rendimiento con la implementación del grafo de escena.

4.1. Diagrama de componente

Este diagrama modela la vista estática del sistema. En él se muestra la organización y las dependencias lógicas entre el conjunto de componentes de software, sean estos componentes de código fuente, bibliotecas, binarios o ejecutables.

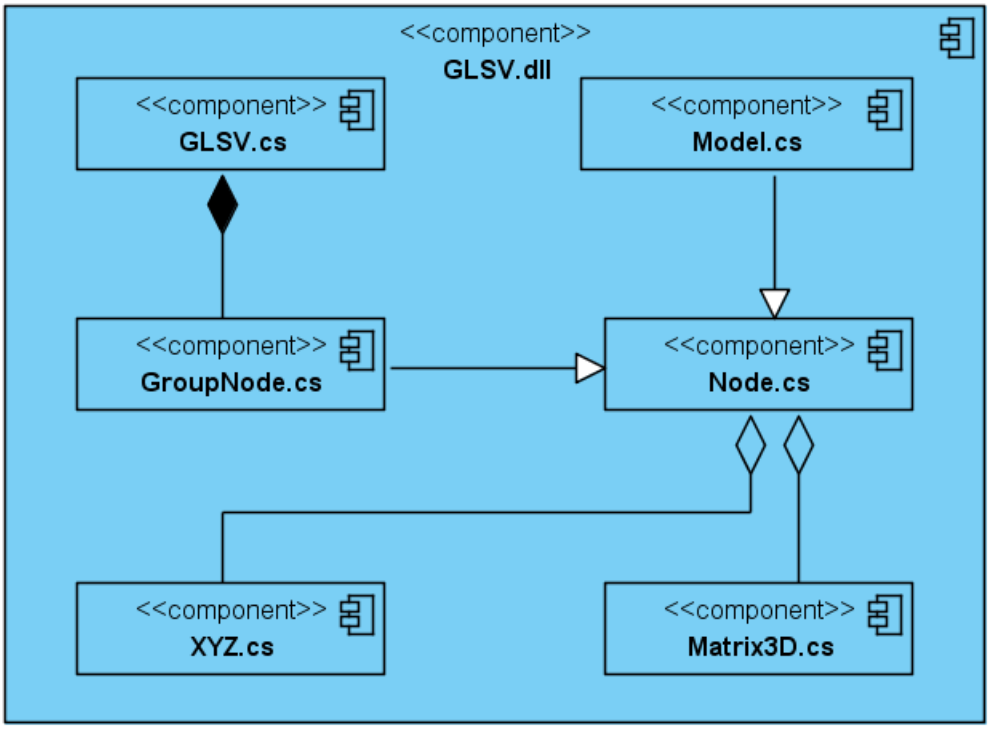

Figura 4.1: *Diagrama de Componentes*

4.2. Validación de los Resultados

La validación de la implementación de la solución propuesta se desarrolló en una computadora con un procesador Intel Core2 Quad Q6600 a una frecuencia de 3.00 GHz, 1 GB de memoria RAM y una tarjeta gráfica ATI Radeon x1550 con 256 MB de RAM para video. Mediante la misma se pretendía evaluar la eficiencia y facilidad en el acceso a los objetos en la escena.

4.3. Datos de Prueba

Los datos de pruebas del grafo de escena fueron tomados teniendo en cuenta dos criterios fundamentales. El primero es la necesidad de implementar una ED jerárquica para la biblioteca GLSVe, por lo que en algunos casos se hicieron pruebas de jerarquías entre diferentes tipos de objetos en la escena. El segundo es comparar la eficiencia en el acceso y la búsqueda de información con relación con la ED que daba soporte a la biblioteca.

4.4. Casos de Prueba

Una vez incorporado el grafo de escena a la GLSVe se cargaron varios prototipos de actividades y se realizaron una serie de pruebas funcionales. A continuación se muestran los resultados que se obtuvieron al ejecutar las funcionalidades incorporadas a la biblioteca con la implementación del grafo de escena.

4.4.1. Funcionalidades incorporadas a la biblioteca GLSVe

Funcionalidades	Lista	Grafo de escena
Adicionar un nuevo elemento	Si	Si
Eliminar un elemento	Si	Si
Buscar un elemento	No	Si
Dibujar la escena	Si	Si
Trasladar uno o varios nodos	No	Si
Rotar uno o varios nodos	No	Si
Escalar uno o varios nodos	No	Si
Cambiar de jerarquía uno o varios nodos	No	Si

Tabla 4.1: *Funcionalidades de GLSVe*

4.4.2. CP 1: Adicionar nodo

Id del escenario	Escenario	Dirección	Nombre	Respuesta del sistema
EC 1	Escenario 1: "Se adicionó el nuevo nodo"	V	V	Adiciona un nuevo nodo a la escena
EC 2	Escenario 2: "El nodo ya existe"	V	V	Muestra un mensaje notificando la existencia del nodo

Tabla 4.2: *Caso de prueba adicionar nodo*

Descripción general: Este CU permite adicionar nuevos nodos al grafo de escena.

Especificaciones del caso de prueba adicionar nodo

Id del escenario	Escenario	Dirección	Nombre	Respuesta del sistema	Resultado de la prueba
EC 1	Escenario 1: "Se adicionó el nuevo nodo"	"mesa.obj"	"mesa"	A un nuevo nodo a la escena	Si no existe el nodo, se adiciona al grafo de escena un nuevo objeto. Si existe el nodo, se muestra un mensaje notificando la existencia del objeto.
EC 2	Escenario 2: "El nodo ya existe"	"mesa.obj"	"mesa"	Muestra un mensaje notificando la existencia del nodo	

Tabla 4.3: *Especificación del caso de prueba adicionar nodo*

4.4.3. CP 2: Eliminar nodo

Id del escenario	Escenario	Nombre	Respuesta del sistema
EC 1	Escenario 1: "Se eliminó un nodo"	V	El sistema busca el nodo y lo elimina.

Tabla 4.4: *Caso de prueba eliminar nodo*

Descripción general: Este CU permite eliminar nodos del grafo de escena.
Condiciones de ejecución: Tiene que haber uno o más nodos en el grafo de escena.

Especificaciones del caso de prueba eliminar nodo

Id del escenario	Escenario	Nombre	Respuesta del sistema	Resultado de la prueba
EC 1	Escenario 1: "Se eliminó un nodo"	"glsv"	El sistema busca el nodo y lo elimina.	Se elimina un nodo del grafo de escena

Tabla 4.5: *Especificación del caso de prueba eliminar nodo*

4.4.4. CP 3: Trasladar nodo

Id del escenario	Escenario	Nombre	Posición	Respuesta del sistema
EC 1	Escenario 1: "Se trasladó un nodo"	V	V	El sistema busca el nodo y lo traslada
EC 2	Escenario 2: "No se encontró el nodo"	V	V	El sistema lanza un mensaje notificando que el nodo no existe

Tabla 4.6: *Caso de prueba trasladar nodo*

Descripción general: Este CU permite trasladar uno o varios nodos en el grafo de escena.
Condiciones de ejecución: Tiene que haber uno o más nodos en el grafo de escena.

Id del escenario	Escenario	Nombre	Posición	Respuesta del sistema	Resultado de la prueba
EC 1	Escenario 1: "Se trasladó un nodo"	"glsv"	(10f,0f,10f)	El sistema busca el nodo y lo traslada.	Se traslada un nodo del grafo de escena.
EC 2	Escenario 2: "No se encontró el nodo"	"glsv"	(10f,0f,10f)	El sistema lanza un mensaje notificando que el nodo no existe	

Tabla 4.7: *Especificación del caso de prueba trasladar nodo*

Id del escenario	Escenario	Nombre	eje	spin	Respuesta del sistema
EC 1	Escenario 1: "Se rotó un nodo"	V	V	V	Busca el nodo y lo rota en el eje y la velocidad especificados.
EC 2	Escenario 2: "No se encontró el nodo"	V	V	V	Lanza un mensaje notificando que el nodo no existe

Tabla 4.8: *Caso de prueba rotar nodo*

Especificaciones del caso de prueba trasladar nodo

4.4.5. CP 4: Rotar nodo

Descripción general: Este CU permite rotar uno o varios nodos en el grafo de escena.
Condiciones de ejecución: Tiene que haber uno o más nodos en el grafo de escena.

Especificaciones del caso de prueba rotar nodo

Id del escenario	Escenario	Nombre	eje	spin	Respuesta del sistema	Resultado de la prueba
EC 1	Escenario 1: "Se rotó un nodo"	"glsv"	(0,1,0)	40	Busca el nodo y lo rota en el eje y la velocidad especificados.	Se rota un nodo del grafo de escena
EC 2	Escenario 2: "No se encontró el nodo"	"glsv"	(0,1,0)	40	Lanza un mensaje notificando que el nodo no existe	

Tabla 4.9: *Especificación del caso de prueba rotar nodo*

4.4.6. CP 5: Escalar nodo

Id del escenario	Escenario	Nombre	Escala	Respuesta del sistema
EC 1	Escenario 1: "Se redimensionó un nodo"	V	V	El sistema busca el nodo y lo redimensiona.
EC 2	Escenario 2: "No se encontró el nodo"	V	V	El sistema lanza un mensaje notificando que el nodo no existe

Tabla 4.10: *Caso de prueba escalar nodo*

Descripción general: Este CU permite redimensionar uno o varios nodos en el grafo de escena.
Condiciones de ejecución: Tiene que haber uno o más nodos en el grafo de escena.

Especificaciones del caso de prueba escalar nodo

Id del escenario	Escenario	Nombre	Escala	Respuesta del sistema	Resultado de la prueba
EC 1	Escenario 1: "Se redimensionó un nodo"	"caja"	(5f,5f,5f)	El sistema busca el nodo y lo redimensiona.	Se cambia de tamaño un nodo del grafo de escena.
EC 2	Escenario 2: "No se encontró el nodo"	"caja"	(5f,5f,5f)	El sistema lanza un mensaje notificando que el nodo no existe	

Tabla 4.11: *Especificación del caso de prueba escalar nodo*

4.4.7. Visualización de la actividad de Agudeza Visual por Transparencia

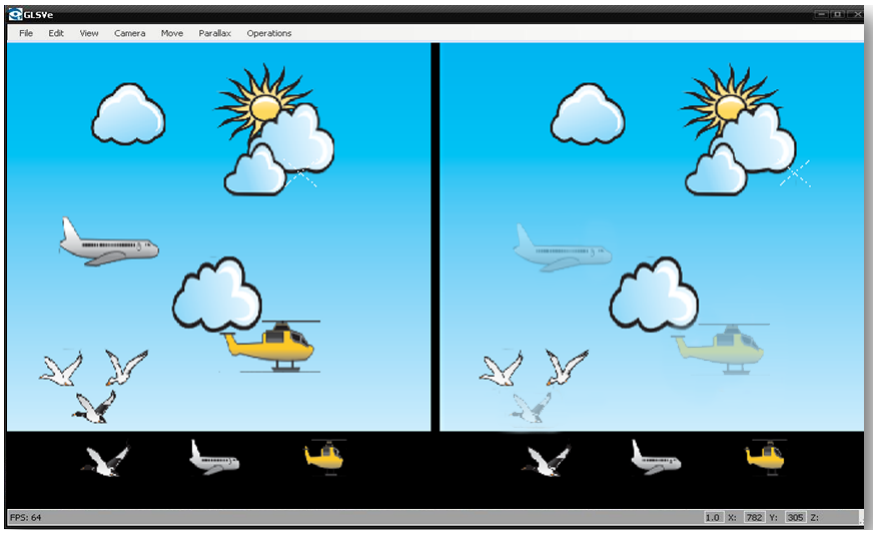

Figura 4.2: *Actividad de transparencia cargada con la lista*

La actividad de Agudeza Visual por Transparencia fue cargada y visualizada mediante la lista y el grafo de escena.

Con la Lista, el acceso a los objetos en la escena se hace un poco difícil a la hora de aplicar alguna transformación en la escena ya sea a uno, o un grupo de objetos. Esto hace que el programador, para acceder a un elemento en la lista, debido a que esta es lineal, tenga que recorrerla desde el principio y aplicar cualquier transformación, elemento por elemento por ejemplo: En el caso de que dicho programador quiera ocultar las imágenes que se muestran en la parte central de la escena, ver figura 4.2 tiene que recorrer la lista, buscarlas una por una, y modificar el atributo de visibilidad, a continuación se muestra un fragmento de código para el acceso a los elementos almacenados en la lista.

```
public void SetVisibility()
{
    for (int i = 0; i < _engine.SceneModels.Count; i++)
    {
        if (((Texture)_engine.SceneModels[i]).Layer == 1)
            ((Texture) _engine.SceneModels[i]).Visible = false;

    }
}
```

Figura 4.3: *Ejemplo de código con lista*

Por otra parte el acceso a los objetos con el grafo de escena es un poco más flexible al programador. Por su estructura de jerarquizados, el grafo de escena permite crear nodos de agrupaciones(ver figura 4.4), permitiendo así aplicar transformaciones en uno o un grupo de objetos. Esto evita que tenga que hacer un recorrido completo de la escena preguntando por los objetos para modificarlos. A continuación se muestra un fragmento de código para el acceso a los elementos almacenados en el grafo de escena.

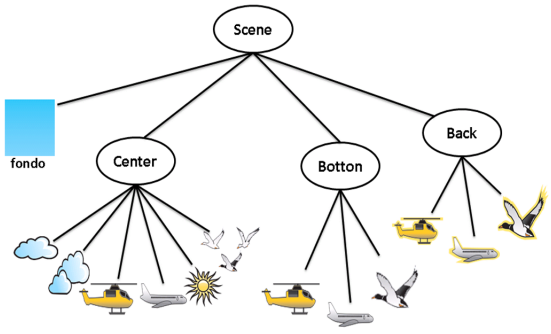

Figura 4.4: *Ejemplo de Actividad de transparencia cargada con el grafo de escena*

Tomando el mismo ejemplo de la visibilidad, en el caso de que dicho programador quiera ocultar las imágenes que se muestran en la parte central de la escena, solo tiene que buscar el nodo de grupo que las contiene y modificar el atributo de visibilidad. Al no dibujarse el nodo, tampoco lo harán sus hijos.

```
private void SetVisibility()
{
     _engine.FindNode("scene").BDisplay = false;
}
```

Figura 4.5: *Ejemplo de código con grafo de escena*

4.4.8. Visualización de otras escenas con el Grafo de escena

Durante las pruebas se decidió cargar y visualizar escenas complejas para probar las funcionalidades implementadas así como el estado jerárquico del grafo.

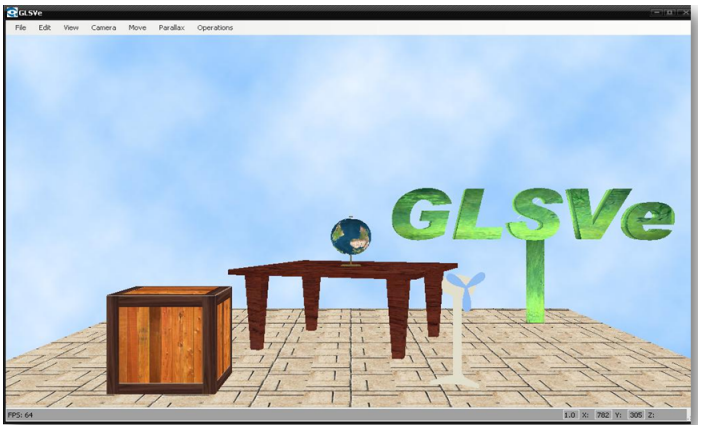

Figura 4.6: *Ejemplo de escena cargada con el grafo de escena*

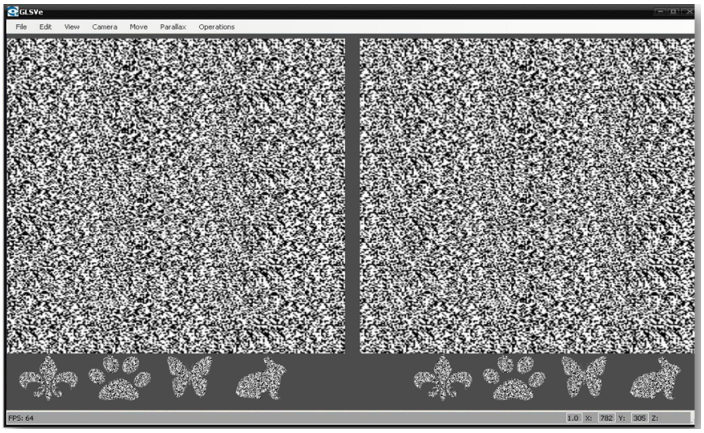

Figura 4.7: *Ejemplo de actividad de vision estereoscópica cargada con el grafo de escena*

Conclusiones

En el transcurso de la investigación realizada se han llegado a las siguientes conclusiones:

- Se logró implementar una estructura de datos que cumpliera con los objetivos trazados para dar solución a la problemática planteada.
- Con el grafo de escena implementado, se logró incorporar a la biblioteca GLSVe nuevas funcionalidades permitiendo establecer una organización lógica y facilitando así el manejo de la escena por parte del programador.
- La biblioteca GLSVe con la implementación de la nueva estructura de datos brinda la posibilidad de cargar escenas complejas.
- Con las funcionalidades implementadas en el grafo de escena se espera lograr un mejor acceso y gestión de los objetos en la escena.

Recomendaciones

Se proponen como recomendaciones las siguientes:

- Se recomienda implementar nodos especializados para mejorar el rendimiento del grafo de escena en la GLSVe.
- Continuar con el estudio de los grafos de escenas para proponer nuevas funcionalidades que permitan mejorar el acceso a los elementos en la escena por parte del programador.

Referencias bibliográficas

[mis, 2006] (2006). Data structure definition. http://www.linfo.org/data_structure.html.

[Mon, 2010] (2010). Realidad virtual. http://www.monografias.com/trabajos11/realitual/realitual.shtml.

[Aaron M. Tenenbaum, 1993] Aaron M. Tenenbaum, Yedidyah Langsam, M. J. A. (1993). *Estructuras de Datos en C*. Prentice Hall Hispanoamerica, S.A., 1ra edition.

[Abreu, 2010] Abreu, L. L. (2010). Sistema de posicionamiento para entornos virtuales. Master's thesis, Universidad de las Ciencias Informáticas.

[Bonanata, 2005] Bonanata, M. (2005). *Programación y Algoritmos*. MP Ediciones S.A., 1ra edition.

[Everly, 2006] Everly, D. (2006). 3d game engine architecture.

[Gutiérrez, 1993] Gutiérrez, X. F. (1993). *Estructuras de datos Especificación, diseño e implementación*. UPC, 1ra edition.

[Junker, 2006] Junker, G. (2006). *Pro OGRE 3D Programming*. Springer-Verlag, 1ra edition.

[Martz, 2007] Martz, P. (2007). Openscenegraph quick start guide.

[Mateo, 2010] Mateo, I. U. (2010). Acceso manual e interfaz gráfica para el juego ai-live. Master's thesis, UNIVERSIDAD CARLOS III DE MADRID.

[Pomier, 2006] Pomier, J. H. T. (2006). Fundamentos de la programación. Master's thesis, Universidad Mayor de San Andrés.

[Román, 2009] Román, Y. R. C. (2009). Optimización de la visualización en la herramienta gráfica scenetoolkit. Master's thesis, Universidad de las Ciencias Informáticas.

[Team, 2006] Team, G. (2006). *The G3D 6.08 Manual*.

Glosario de Términos

Biblioteca: desde el punto de vista informático es una colección o conjunto de subprogramas usados para desarrollar un software.

Realidad virtual: Representación de escenas u objetos producidos por un sistema informático, dando la sensación de su existencia real.

Tao Framework: Interfaz de programación de C♯ para acceso a bibliotecas escritas en C++.

Visión estereoscópica: Visión binocular.

Estructura de Datos: En informática una estructura de datos es una forma de organizar un conjunto de datos elementales.

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

Motor Gráfico: (graphic engine en inglés) módulo gráfico independiente, lo bastante potente para poder tratar toda la información que hay en él, así como su visualización en tiempo real. Este concepto surge por la complejidad gráfica añadida a la hora de tratar con escenarios 3D.

Acrónimos

GLSVe Graphics Library for Stereoscopic Vision engine

HDSVE Herramientas de Desarrollo para Sistemas de Visión Estereoscópica

RV Realidad Virtual

VE Visión estereoscópica

3d Tres dimensiones

ED Estructura de Datos