**UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS**

**Facultad 5 ENTORNOS VIRTUALES**

# ALGORITMOS PARA LA MANIPULACIÓN EFICIENTE DE OBJETOS DINÁMICOS EN ESCENAS VIRTUALES URBANAS

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE
"INGENIERO EN CIENCIAS INFORMÀTICAS"

**Autores:** Mariela Nogueira Collazo

Omar Correa Madrigal

**Tutores:** Dr. José Ignacio Guzmán Montoto.

Ing. Yanoski Camacho Román.

Ciudad de la Habana

Mayo - 2007

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas (UCI) para que haga el uso que estimen pertinente con este trabajo.

Para que así conste firmamos la presente
A los ____ días del mes de _____ del año _____.

**Mariela Nogueira Collazo** **Omar Correa Madrigal**

_____ _____
Firma del Autor Firma del Autor

**Yanoski Rogelio Camacho** **José I Guzmán Montoto**

_____ _____
Firma del Tutor Firma del Tutor

# Dedicatoria

*A mis padres.*

*A Meri.*

*A mi abuela donde quiera que esté.*

Mariela.

*A mi hermano*

*A mi novia.*

*A mis padres por apoyarme tanto durante estos largos años de universidad.*

Omar.

# Agradecimientos

*A mi madre, por su apoyo incondicional.*

*A mi padre,  por ayudarme a mantener la mente positiva.*

*A Yaima y Marilis, por sus buenos consejos.*

*A todos mis compañeros y profesores que de una forma u otra hicieron posible este trabajo.*

*A todos ustedes, gracias.*

Mariela.

A todos los compañeros y profesores que colaboraron conmigo.

Omar.

## Resumen

El desarrollo de la Realidad Virtual en la actualidad ha abierto paso a nuevos campos investigativos con vista a aumentar las potencialidades de estos sistemas. La interactividad y el realismo de las escenas simuladas no siempre alcanzan la calidad requerida, y es precisamente este, uno de los principales problemas que enfrentan hoy estas aplicaciones.

Esta tesis aborda el desarrollo de dos algoritmos: el Algoritmo de los TBV y el Algoritmo de Reubicación, que tienen como objetivo emplear eficientemente los recursos de hardware durante la manipulación de objetos dinámicos en entornos virtuales urbanos. También se define una propuesta de diseño de un módulo de clases para incluir estos algoritmos a la herramienta *SceneToolKit*.

Los resultados obtenidos en esta investigación se muestran a través de diferentes pruebas realizadas que dan una medida clara de las potencialidades de los algoritmos implementados.

# Índice

# Introducción

La realidad virtual se desarrolla vertiginosamente en nuestros días, esto ha dado paso a la incorporación de novedosas técnicas que han potenciado la modelación 3D de alta calidad, la confección de complejos modelos matemáticos capaces de simular sucesos del mundo real, la creación de algoritmos de inteligencia artificial para crear convivencia de objetos dinámicos, entre otros elementos que, sin duda, han propiciado el incremento de la interactividad y el realismo en los entornos virtuales.

Al mismo tiempo, estos elementos implican que a la hora de poner en marcha una aplicación de realidad virtual, puedan aparecer dificultades que obstaculicen su procesamiento en tiempo real, como son: la carencia de una visualización fluida, la insuficiente velocidad de cómputo y la poca disponibilidad de almacenamiento.

En la UCI (Universidad de las Ciencias Informáticas), existen proyectos que desarrollan software de Realidad Virtual. Estos grupos de trabajo se dedican a la creación de Simuladores como el de Conducción de Auto desde el año 2003 hasta la actualidad, para esto utilizan una Herramienta “SceneToolkit” que permite el desarrollo de estos entornos virtuales. Dicha Herramienta necesita de algoritmos que optimicen el uso de los recursos (memoria, velocidad de cómputo y visualización), en tiempo de ejecución, de manera que se produzca una simulación fluida (preferiblemente de 60 Hz) que permita la credibilidad del proceso simulado.

La Herramienta no cuenta con un módulo de optimización que garantice una manipulación eficiente de los objetos dinámicos de las escenas. Este hecho atenta contra la visualización de escenas representativas del mundo real donde aparecen elementos en movimiento.

Ante estos inconvenientes surge la interrogante: **¿Cómo emplear eficientemente los recursos de hardware durante la manipulación de objetos dinámicos en los entornos urbanos?**

La presencia de objetos dinámicos en los escenarios virtuales, si bien le da mucho más realismo y vida a estos sistemas, crea una demanda mayor de cálculos y recursos, por tanto, el problema a resolver en esta tesis, es la ausencia de algoritmos capaces de manejar eficientemente los objetos dinámicos y reducir el consumo de recursos, dando una mayor sensación de actividad a la escena.

El objeto de estudio del presente trabajo son los **algoritmos que reducen el consumo de recursos durante el proceso de visualización de los entornos virtuales en los sistemas de realidad virtual.**

Para el desarrollo de este trabajo se definió como **campo de acción**: los algoritmos y técnicas empleados para disminuir el consumo de recursos que provoca el trabajo con objetos dinámicos en entornos virtuales de ciudades donde la cámara virtual se traslada siguiendo la superficie.

Este proyecto se propone como objetivo general: **Implementar algoritmos que permitan el manejo eficiente de los objetos dinámicos en los entornos urbanos.**

**Como objetivos específicos se plantean:**

1. Obtener una solución para la selección de visibilidad de los objetos dinámicos.
2. Obtención de una solución para la reubicación de los objetos dinámicos.
3. Proponer el Diseño de un Módulo para la Manipulación Eficiente de los Objetos dinámicos en la Herramienta “Scene Toolkit”

Para dar cumplimiento al objetivo se plantean las siguientes tareas:

- Investigación de las principales características de los algoritmos que logran manejar eficientemente los elementos dinámicos en escenas virtuales.

- Desarrollo de soluciones técnicas basadas en los algoritmos estudiados.

- Implementación de los Algoritmos que garanticen la manipulación eficiente de los objetos dinámicos en las escenas virtuales urbanas.

- Estudio de la Arquitectura de la Herramienta “SceneToolkit”.

- Diseño de un módulo de clases para lograr la Manipulación de los Objetos Dinámicos en la Herramienta

# Capítulo 1 Fundamentación Teórica

## Introducción

La visualización eficiente de las escenas dinámicas compuestas por varios millones de polígonos, es uno de los problemas más desafiantes del gráfico por computadora hoy día, para lograr una fluidez adecuada se emplean algoritmos que manipulan eficientemente los objetos dinámicos del entorno y reducen el consumo de recursos.

En este capítulo se hace una breve introducción a las técnicas básicas sobre las cuales se fundamentan estos algoritmos. Los más eficientes empleados en entornos dinámicos tienen como características principales: que son *sensibles a la salida*, interactivos, poseen técnicas de subdivisión del espacio que aseguran un eficiente almacenamiento de los componentes del entorno y permiten una rápida actualización de sus estructuras de datos en consecuencia con la dinámica de la escena [2].

## 1.1 Coherencia Espacio Objeto

Esta técnica explota la coherencia en la localización tridimensional de los objetos. Puede entenderse esto como el sistema de coordenada universal en el cual coexisten los elementos del entorno virtual y donde se le realizan transformaciones básicas. [7] Existen variados algoritmos que implementan esta técnica mediante diferentes estructuras de datos y que muestran la diversidad que existe en cuanto a formas de organizar los objetos de una escena. En este epígrafe se muestran algunas de las estructuras de datos más empleadas.

### 1.1.1 Estructuras Jerárquicas

#### 1.1.1.1 Bounding Volume Hierarchy

La técnica de jerarquía de volúmenes frontera fue creada por Steven Rubin y Turner Whitted (1980). Se basa en encerrar los objetos de una escena en un *bounding volume* (volumen de encierro que puede ser cuadrado, esférico, etc.) Y se organizan en una estructura jerárquica los múltiples *bounding*. Véase figura 1. Los rectángulos grises son los *bounding volume* de los objetos y los otros rectángulos son los *bounding volume* que forman la jerarquía.

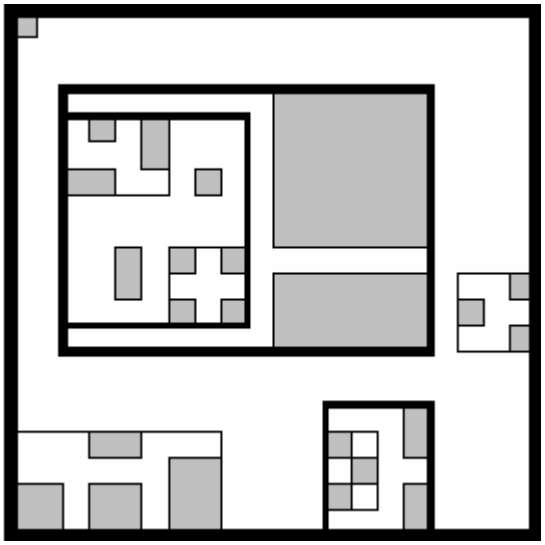

**Figura 1 Jerarquía de volúmenes frontera**

Este tipo de estructura es apropiada para escenas con objetos dinámicos, pero la reubicación de la jerarquía de los objetos que se trasladan en la escena puede resultar complicada. Además, como muestra la figura no es nada fácil construir automáticamente una buena jerarquía, de hecho esto constituye un campo de investigación abierto. [8]

Existen varios tipos de volúmenes fronteras, que en dependencia de las características de los objetos de la escena son más o menos adecuados, uno de los más empleados es el *Bounding Spheres* o esferas fronteras. Sin embargo, cuando se tiene un objeto irregular, a menudo la esfera frontera ocupa un gran espacio en el que el objeto realmente no existe, lo que puede representar una deficiencia para algunos algoritmos. Para resolver este problema, se emplean entonces los *Bounding Boxes* o cajas fronteras, que se determinan buscando los mayores y menores vértices en relación con el centro del objeto, hallándose una caja que contiene al objeto [2].

En dependencia de la aplicación que se esté haciendo, se pueden usar ambas técnicas como pruebas iniciales. Se puede usar una jerarquía de pequeñas esferas y cajas en el objeto. [8]

Otros volumenes fronteras utilizados son: cápsulas, *lozenges* (pastilla, gragea), cilindros y elipsoides. [2]

#### 1.1.1.2 BSP-Tree

El árbol de partición binaria del espacio (*Binary Space Partitioning-Tree) BSP-Tree* basa su estructura en un árbol binario en el cual cada nodo representa un plano. El subárbol izquierdo corresponde al medio espacio negativo y el derecho al positivo. La siguiente figura da muestra de ello. Las líneas denotan los planos y son representados por letras y las regiones por números.

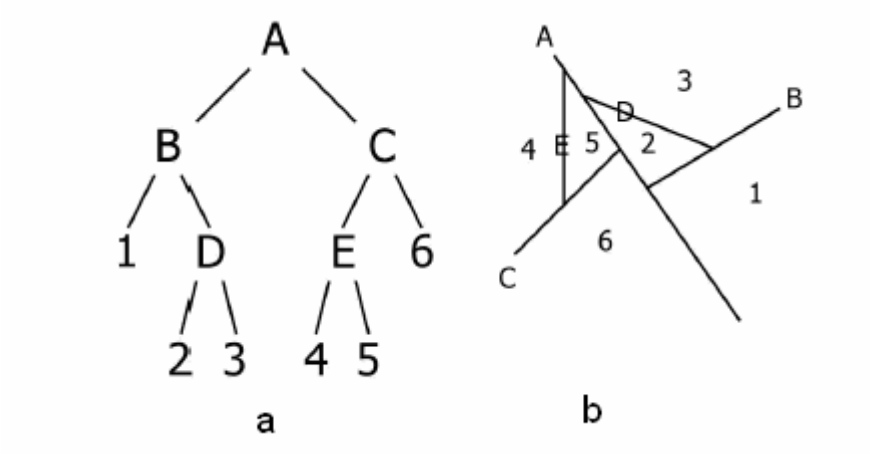


**Figura 2 La figura (a) representa el BSP-Tree correspondiente a la representación espacial (b)**

Esta técnica dada su estructura permite realizar grandes optimizaciones utilizando las potencialidades de la oclusión, de hecho fue esa una de las causas de su surgimiento.

La construcción de un modelo *BSP- Tree* a partir de otro modelo, resulta muy complicado y es esta precisamente una debilidad de esta estructura. La mayoría de los modelos están determinados en otros formatos, como la representación por volúmenes frontera o *bounding volumen*. En este caso por ejemplo, el proceso de conversión se hace difícil. Las técnicas conocidas para estas trasformaciones no son *sensibles a la salida*. Además, el *BSP-Tree* asume que los *bounding volumen* son polígonos y que sus normales van dirigidas hacia fuera del objeto [8], pero como Sundarsky y Gotsman apuntan, actualmente sólo algunos modelos presentan estos requisitos. [6]

#### 1.1.1.3 Octree

Esta estructura fue introducida por Andrew Glassner en 1984 para la subdivisión del espacio objeto. Para construir un octree primeramente se inicializa un *voxel* que encierre todas las geometrías presentes en la escena, después se subdivide recursivamente cada *voxel* que encierre muchas geometrías en ocho subvoxels (octantes) de igual tamaño. La Figura 3 representa estas subdivisiones. La

condición de parada de las subdivisiones puede estar dada por la cantidad de polígonos o la cantidad de objetos.

En general, la subdivisión de los espacios mediante octree se basa en colocar varios *voxels* pequeños en los sitios donde hay muchas geometrías y por el contrario, en los lugares que contengan pequeñas y pocas geometrías se utilizan pocos y grandes *voxels*.

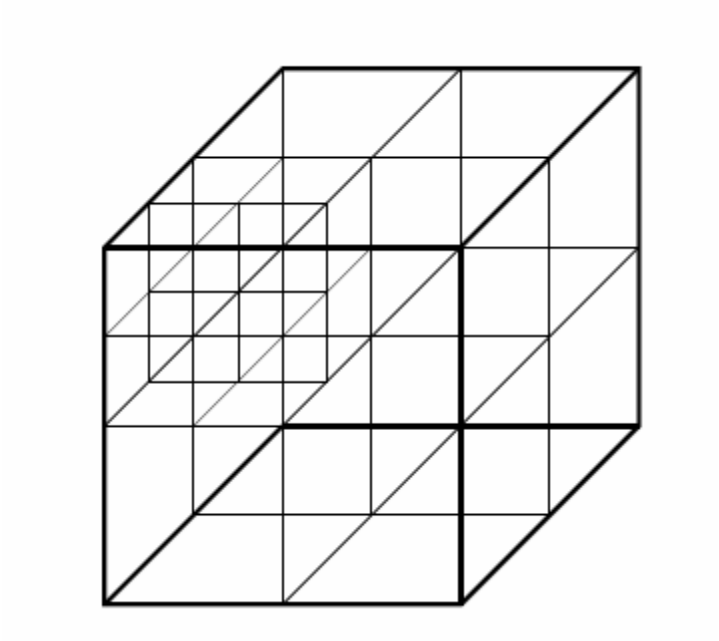

**Figura 3 Representación visual del Octree.**

Esta técnica no es muy beneficiosa para tratar escenas dinámicas pues la actualización de la estructura de datos no es trivial como en otros casos, especialmente si es necesario reconstruir el octree sobre la marcha, lo cual sucede muy a menudo.

Existen formas de disminuir la desventaja que trae consigo esta reconstrucción del árbol, como es el caso la técnica Mínimo Común Antecesor (*Least Common Ancestro)* LCA con la cual se obtienen resultados muy alentadores pero aun así en escenas dinámicas muy cargadas los beneficios son ínfimos. [8]

## 1.1.2 Estructuras No Jerárquicas

### 1.1.2.1 Rejilla Regular

Esta estructura explota el espacio objeto realizando una división regular del espacio como se muestra en la Figura 4.

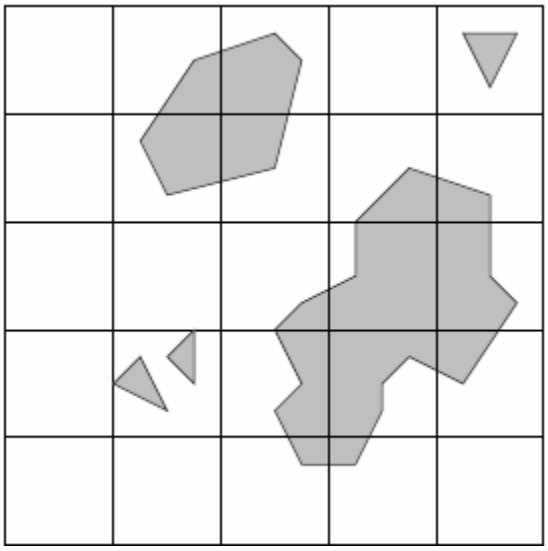

**Figura 4 Rejilla Regular**

Las Rejillas Regulares son apropiadas para escenas dinámicas cuando se combinan con los *bounding volumen* para los objetos, porque reubicar o relocalizar las geometrías resulta fácil: basta con sustraerla de la celda actual y colocarla en la nueva.

Una debilidad de esta estructura radica en que manejan áreas densas y escasas de objetos, con el mismo tamaño de subdivisión. Esto implica que no se puedan desechar extensas partes del modelo, cosa que con las estructuras jerárquicas si es posible; ejemplo: considere una escena de una cuidad, aquí puede que una celda encierre una tienda y por tanto contendrá muchas geometrías, mientras que otra celda encierra solo la porción de una calle, o sea, un pequeño número de geometrías. Se debe tratar de subdividir el espacio en

celdas que contengan la mayor cantidad posible de geometrías, porque esto ayuda a que la manipulación de las mismas sea menos costosa [8].

## 1.2 Manipulación de la información espacial del entorno 3D.

En las escenas 3D donde existen objetos dinámicos es muy conveniente conocer la información referente a la ubicación espacial que tiene cada objeto en el entorno, esto permite controlar su movimiento.

A partir de estudios realizados por programadores de los grupos desarrolladores de Sistemas de Realidad Virtual de la Facultad 5 se obtuvo una herramienta de software que permiten la transición de un modelo tridimensional de ciudad creado en el editor gráfico 3D Studio Max, a un fichero de extensión .grf que contiene información de las posiciones espaciales de todas las trayectorias existentes en un entorno 3D por donde se pueden trasladar los objetos dinámicos. Conociendo esta información es fácil controlar el movimiento de estos objetos, ya que se puede saber su posición en cada instante de su trayectoria.

Para procesar la información contenida en el fichero .grf se utiliza como estructura de datos: un grafo (*grafo de trayectorias del entorno*).

El *grafo de trayectorias del entorno* está compuesto por un conjunto de *aristas* (links) y *nodos*, las aristas representan los caminos del entorno, y los nodos las intercepciones entre los caminos. Ver Figura 5

La información almacenada en el *grafo de trayectorias del entorno* es a ciencia cierta un conjunto de vectores 3D que marcan las posiciones reales de las trayectorias del entorno. Cada *link* del Grafo es almacenado como una polilínea

la cual representa un conjunto de vectores 3d que marcan la línea central de la trayectoria, por ejemplo: una carretera por donde circulan los autos del entorno.

Los nodos del *grafo de trayectorias del entorno* representan las intercepciones entre la trayectorias del entorno, constituyen puntos de toma de decisión para los objetos ya que cuando un objeto llegue a un nodo debe decidir qué nueva trayectoria va a seguir, pues de un nodo parten varias aristas.

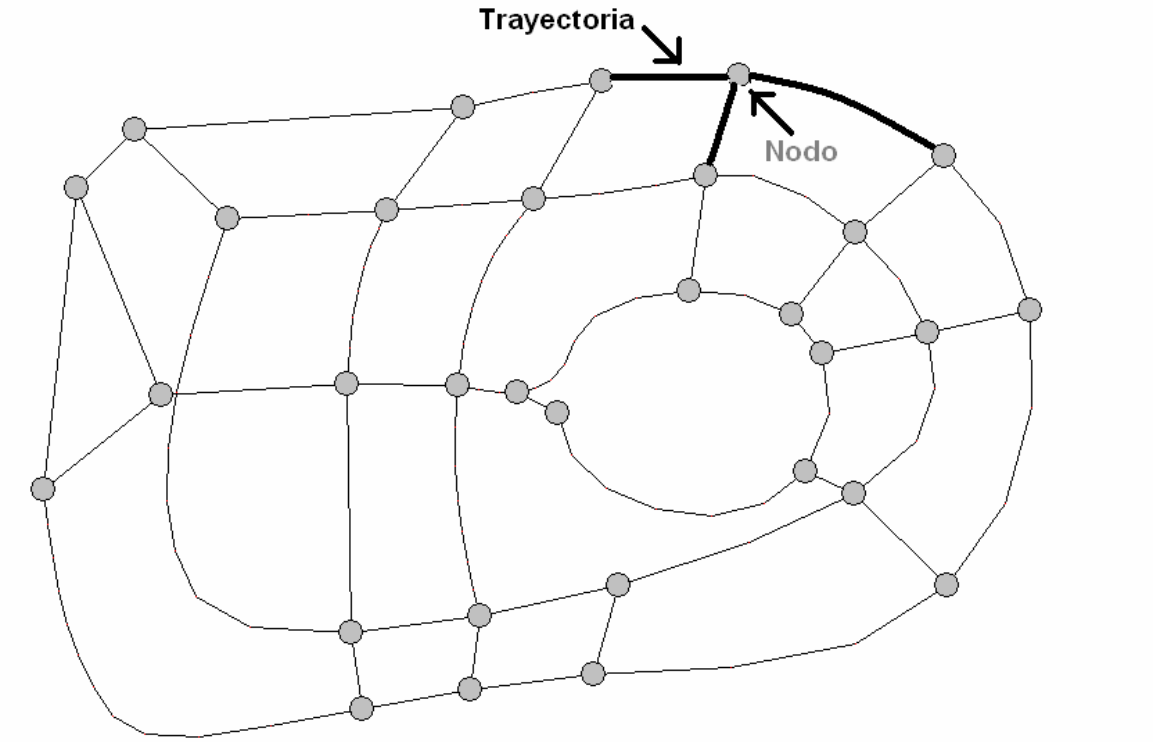


**Figura 5 Representación del Grafo de Trayectorias del Entorno.**

### 1.2.1 Aspectos de la traslación de los objetos por el entorno

Para efectuar la traslación de los objetos dinámicos por el mundo 3d se utiliza la información proveniente del fichero .grf que está almacenada en el *grafo de trayectorias del entorno.* Un entorno virtual puede tener más de un *grafo*, por ejemplo: puede haber un *grafo de trayectorias del entorno* que especifique el conjunto de trayectorias aéreas por donde se pueden trasladar aviones en el

mundo virtual y también un grafo que especifique las carreteras del entorno transitables para los autos y así para cada tipo específico de objeto dinámico.

Cada objeto es asociado al *grafo de trayectorias del entorno* que contiene las trayectorias accesibles para él. A partir de la información que está almacenada en este grafo, se le calcula al objeto la posición que le corresponde en el entorno 3d en cada instante de su traslación, permitiendo así un movimiento continuo del objeto durante la visualización de la escena. Para realizar este cálculo se localiza la arista (Link) del grafo en la que se encuentra el objeto y específicamente el vector 3d que marca su posición dentro de esta arista, teniendo en cuenta el desplazamiento del objeto se efectúan operaciones entre vectores para calcular su siguiente posición.

Las aristas del grafo marcan la línea centro de las trayectorias del entorno, la traslación de los objeto en algunos de los casos no se hace siguiendo esta línea centro, por ejemplo, en el caso de los autos, ellos no se mueven por el centro de la carretera sino por las sendas laterales (derecha o izquierda) o sea que, la línea por donde se traslada el objeto tiene un desplazamiento lateral con respecto a la arista del grafo. En estos casos para efectuar el movimiento de los objetos, es una premisa calcular las coordenadas de cada vértice de la arista desplazado lateralmente, para esto se utilizan propiedades de los vectores ortogonales, de esta manera se va conformando la nueva polilínea (desplazada) en la que se va moviendo el objeto. Ver Figura 6.

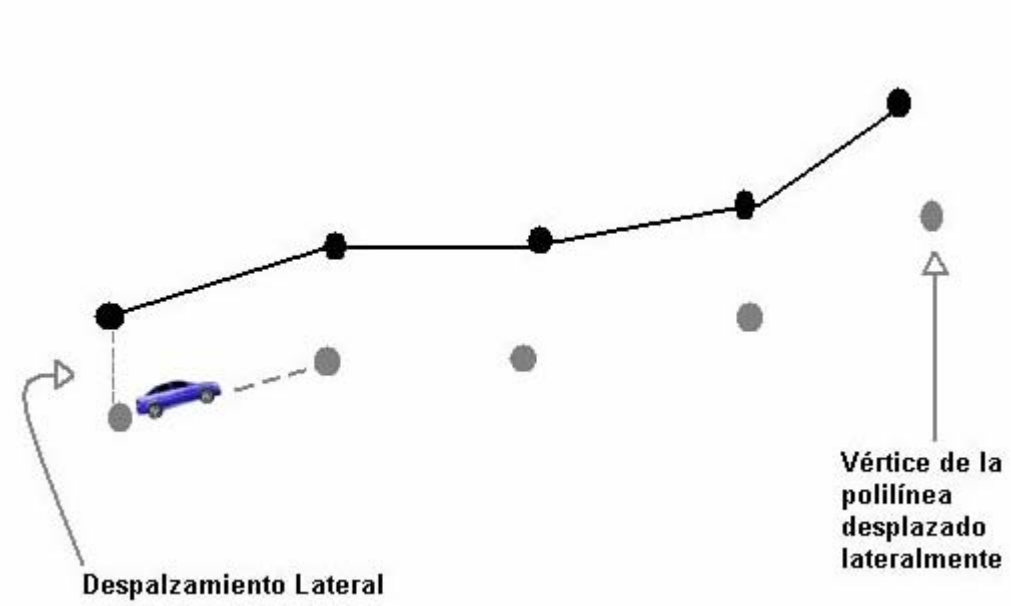


**Figura 6 Desplazamiento lateral que pueden tener los objetos en la escena.**

En la Figura 6 se muestra una polilínea de un grafo de trayectorias del entorno que representa las carreteras del mundo virtual por las que se pueden trasladar los autos. La polilínea (link del grafo) tiene señalado cada uno de sus vértices (vectores 3d) y el desplazamiento lateral que define una de las sendas laterales de la carretera representada. Para efectuar el movimiento del objeto se calcula cada uno de los vértices de esta polilínea desplazados y a partir de ellos se obtiene la nueva trayectoria por la que se va a mover el objeto dinámico, en este caso un auto.

## 1.3 Volumen de Visión (View Frustum)

### 1.3.1 View Frustum 3D

El *View Frustum* 3D en la visión en perspectiva es una pirámide truncada que representa una sección 3D visible por la cámara. Cualquier objeto fuera del volumen de visión no será visto por la cámara, y por tanto no se tendrá en cuenta en la proyección.[2] Está formado por 6 planos que lo delimitan como se

muestra en la Figura 7, llamados cerca, lejos, arriba, abajo, izquierda y derecha. [5]

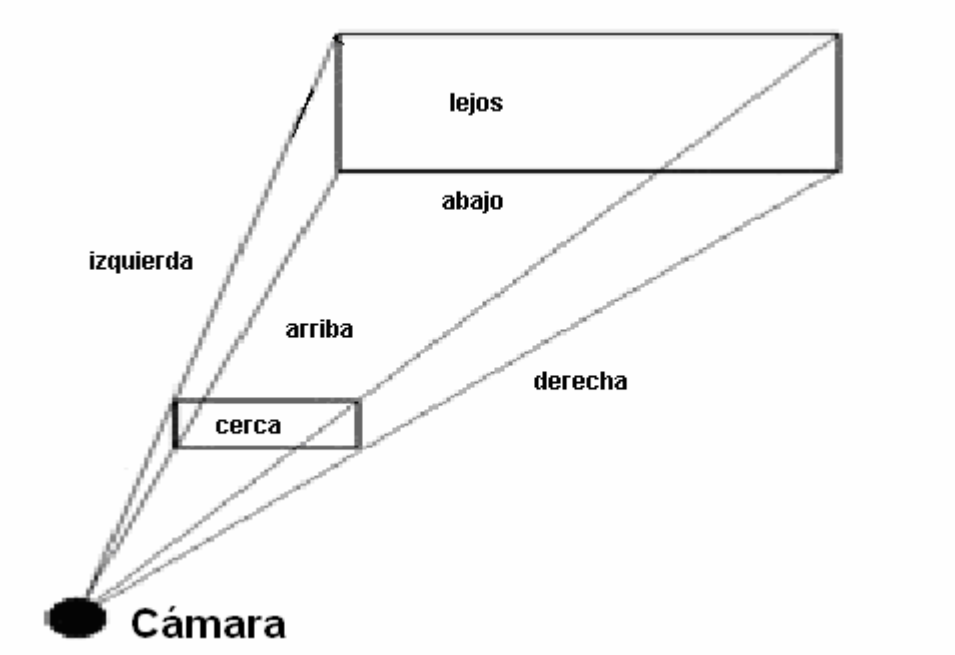


**Figura 7 View Frustum 3D.**

### 1.3.2 View Frustum 2D

El *View Frustum* 2D es el resultado de la proyección del View Frustum 3D sobre el plano XY. Este *frustum* está delimitado por rectas como se muestra en la Figura 10

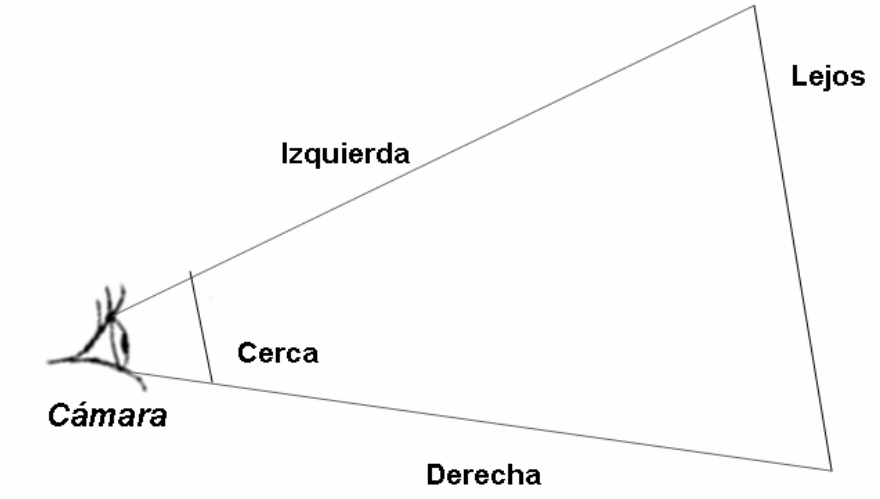


**Figura 8 View Frustum 2D.**

Este no es tan exacto como el *View Frustum* 3D pero sí impone un menor costo computacional a la hora de determinar cuáles objetos de la escena son visibles. Ambos pueden ser ventajosos en dependencia del tipo de entorno a visualizar. [1]

## 1.4 Volúmenes de Fronteras Temporales (TBV) (Temporary Bounding Volumen)

Sudarsky y Gorsman introducen en su artículo "Dynamic Scene Occlusion Culling" de 1999 la utilización de los *TBV* para manipular los objetos dinámicos de una escena. Forma parte de las técnicas que explotan la Coherencia Temporal [7]. Esta técnica parte de las siguientes problemáticas: en primer lugar construir las estructuras toma mucho tiempo, es mejor actualizarlas a partir del movimiento de los objetos dinámicos. Pero si esta actualización se aplica a todos los objetos en movimiento incluso los no visibles, el algoritmo no será *sensible a la salida*. Una alternativa es ignorar a los objetos que no son visibles, pero no se pueden olvidar completamente porque quizás sean visibles en otro momento. [8]

Este algoritmo se basa en realizar una "evaluación perezosa" de los objetos en movimiento, los datos obtenidos son almacenados en un *TBV*. Un *TBV* para

objetos dinámicos garantiza contener a estos durante un período de tiempo. Este período de tiempo es llamado “período de validez” del *TBV* y al final de este período se le llama “vencimiento de la estructura” [6]. Ver Figura 9.

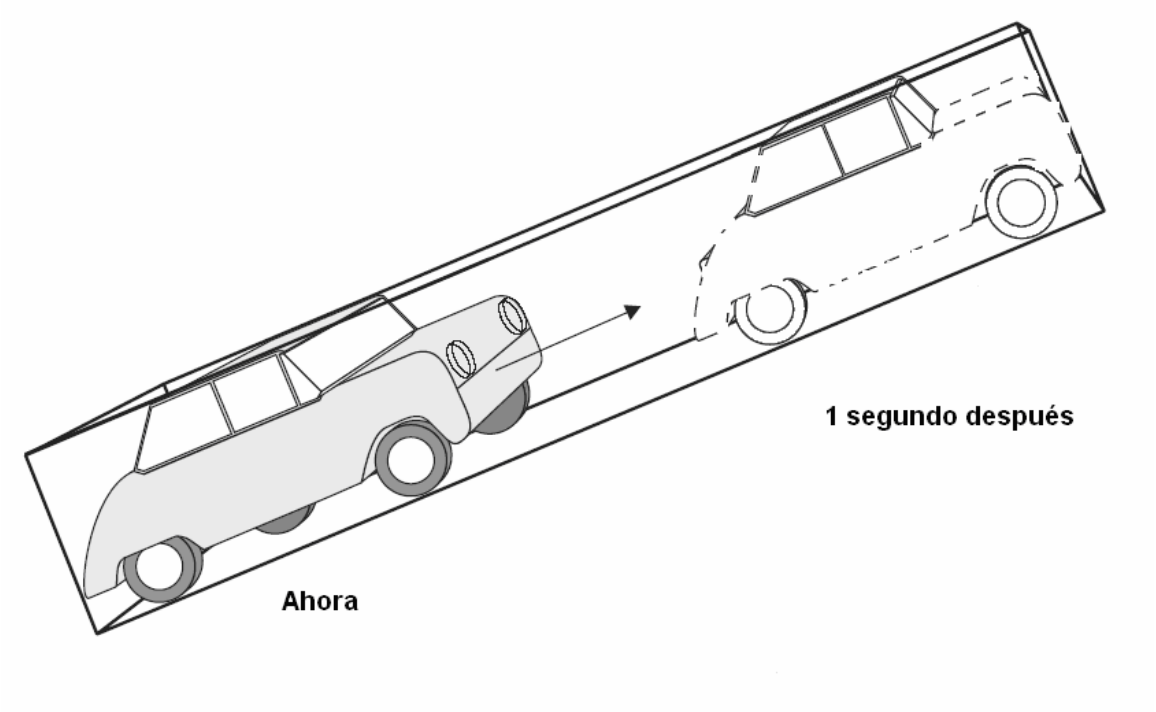


**Figura 9 TBV con un período de validez de 1 s.**

Para construir el *TBV* para un objeto se necesita algunos conocimientos sobre su movimiento. Afortunadamente muchos objetos tienen algunas restricciones que se pueden usar. Si sólo el máximo de velocidad o el máximo de aceleración son conocidos el *TBV* puede ser construido. Estos son construidos en tiempo de ejecución y en aplicaciones donde el flujo de eventos es desconocido de antemano.

En general la principal ventaja de este método radica en que es compatible con otras técnicas y en la mayoría de los casos, resulta muy beneficioso por su alta sensibilidad a la salida. [3]

### 1.4.1 Técnica del TBV utilizando el BSP-Tree

El algoritmo del *BSP-Tree* tiene un alto costo en comparación con otros algoritmos de visibilidad debido a la cantidad de cálculos numéricos que involucra. Este método es mucho más lento que el algoritmo el *Z-buffer* jerárquico. Sin embargo es interesante ver como este algoritmo mejora cuando se le incorpora la técnica de los *TBV*.

Los *BSP-Tree* en su esencia son diferentes de los Octree, ellos representan los objetos mismos, mientras que el octree es meramente una estructura de datos auxiliar, por lo que el algoritmo *BSP-Tree* generalizado a escenas dinámicas es un tanto diferente del basado en *Octrees*.

La técnica del Naylor *BSP-Tree* [3] presenta numerosas ventajas sobre el Z-buffer jerárquico, particularmente la independencia del hardware gráfico y del tamaño de la imagen. Por lo cual a la hora de vincular los *TBV* con el *BSP-Tree* es recomendado utilizar este método. [6]

### 1.4.2 Técnica de los TBV utilizando el Octree

Para escenas dinámicas el octree necesita ser modificado en tiempo real. Para insertar un objeto (*TBV*) en el árbol, las primitivas del objeto son asociadas con los nodos que este intercepta, y los nodos que están asociados con demasiadas primitivas son subdivididos. Para borrar un objeto o *TBV*, sus primitivas son desvinculadas de los nodos a los cuales estaban asociados, entonces se acoplan los nodos asociados con las mismas primitivas, o con un mínimo suficiente de estas.

A la hora de actualizar el *Octree* para el movimiento de objetos dinámicos, el objeto puede ser eliminado del árbol y entonces insertado en la nueva posición.

Si la actualización es realizada para un número relativamente pequeño de objetos dinámicos será mucho más eficiente que si se reconstruye el árbol completo. De lo contrario el algoritmo no es óptimo pues se emplea tiempo innecesariamente acoplando nodos que serán inmediatamente subdivididos otra vez.

Una mejoría para este algoritmo la aporta el empleo de la técnica del padre inmediato superior (*LCA*), que elimina la necesidad de actualizar el octree completo, sólo es necesario hacerlo para el subárbol que contiene al padre inmediato superior del objeto.[8]

El método del octree implementado con *LCA* al unirse con los *TBV* conforma un algoritmo más óptimo gracias a que el *TBV* disminuye la cantidad de objetos a actualizar en cada *frame* de la escena.[6]

## 1.5 Selección de Visibilidad mediante Rejilla Regular

La Rejilla Regular representa una discretización del espacio donde cada celda identifica las características locales de la escena.

Existen propuestas que promueven el debate del uso de las Rejillas Regulares dentro de entornos dinámicos. H.C.Batagelo y Wu Shin-Ting en "*Dynamic Scene Occlusion Culling using a Regular Grid*" [4] proponen el uso de la oclusión y de los *TBV* para resolver los problemas de los entornos dinámicos realizando para ello cinco etapas.

Para el desarrollo de estas etapas ellos se apoyan en cuatro matrices que representan los atributos de los *voxels* o *celdas*:

- La matriz de oclusión ( ) clasifica cada *voxel* como opaco o no opaco. Un *voxel* es opaco si está totalmente incluido dentro de los objetos potencialmente visibles.

- La matriz de oclusión ($\mathcal{H}$) clasifica cada *voxel* como ocluido o no ocluido. Un *voxel* es totalmente ocluido si está oculto detrás de un *voxel* opaco o *voxel*s ocluidos.
- La matriz de identificación ($\mathcal{I}$) asocia una lista de identificadores de objetos (Identificadores de objetos) a cada *voxel*. Estos objetos abarcan la región espacial de ese *voxel*.
- La matriz de *TBV* ($\mathcal{T}$) asocia una lista de identificadores de *TBV* a cada *voxel* no visible.

En cada *frame* se efectúan los siguientes pasos:

- **Discretización de la escena**: Se actualiza la Rejilla Regular para los objetos que forman parte de ***PVS*** *(Potentially Visible Set)* del *frame* anterior, usando las matrices $\mathcal{O}$ y $\mathcal{I}$, y se manipulan los objetos no visibles de acuerdo a sus *TBV* utilizando la matriz $\mathcal{T}$.
- **Transversal View-Frustum:** Recorrer las celdas contenidas dentro del *View-Frustum* para detectar los objetos potencialmente visibles así como los *voxels* que pueden ser usados como oclusores.
- **Extensión de Oclusores**: Extender cada oclusor buscando durante el recorrido dentro del *View-Frustum* las celdas adyacentes no ocluidas.
- **Cálculo de la Oclusión**: Calcular un volumen de oclusión para cada oclusor extendido y calcular las celdas ocluidas.

Este algoritmo es aplicable para cualquier tipo de escena independientemente de la naturaleza de los objetos que la conforman y realiza un análisis muy riguroso de la oclusión. Permite además una eficiente manipulación de los objetos dinámicos gracias a la utilización de técnicas como la combinación de la Rejilla Regular y los *TBV*.

El análisis de la oclusión se hace *frame* a *frame*, con el objetivo de analizar no sólo a los objetos estáticos como posibles oclusores sino también a los dinámicos. Esto resulta muy beneficioso para entornos donde los objetos

dinámicos son geometrías grandes y en consecuencia ocluyan extensas áreas de la escena.

En el caso específico de los entornos urbanos, donde no aparezcan aglomeraciones de objetos dinámicos por encima de la línea de visibilidad este análisis *frame* a *frame* resulta innecesario y costoso porque los objetos dinámicos nunca llegan a ocupar grandes espacios en la escena, por lo cual, realizar el tratamiento de los objetos dinámicos como posibles oclusores no es provechoso, por el contrario, resulta más costoso hacerlo que dibujar aquellas geometrías que puedan ser ocluidas (total o parcialmente) por estos. Además, en esos entornos es muy importante reducir el tiempo de cálculo de visibilidad durante la simulación, lo cual se obstaculiza con el procesamiento *frame* a *frame*.

## 1.6 Biblioteca dPVS

dPVS es un producto creado por la empresa “HYBRID Graphics” líder en la optimización de visibilidad. Esta se dedica a brindar soporte gráfico tanto 2D cómo 3D a consumidores electrónicos. [11]

dPVS es una avanzada herramienta de optimización de visibilidad para desarrollar juegos con extensos y dinámicos mundos 3D [10], está implementada en C++ [7]. Realiza los cálculos de visibilidad en tiempo real, habilitando la creación de mundos 3D interactivos y modificables. Provee soluciones muy eficientes para la optimización de la visibilidad que no requieren preprocesamiento y manipulan entornos dinámicos de cualquier estructura tipológica. También emplean provechosas técnicas de visibilidad como son selección de visibilidad jerárquico (*hierarchical view frustum culling)* y selección de visibilidad por oclusión (*occlusion culling).*

Esta Biblioteca brinda soluciones *sensibles a la salida* y que explotan eficientemente las potencialidades de optimización que soportan las tarjetas gráficas [7]. Está habilitada para computadoras y videoconsolas como *XBOX* y *PlayStation*. [10]

Evidentemente un Sistema de Realidad Virtual que utilice esta librería obtendrá buenos resultados en el manejo de los objetos dinámicos. Por tal motivo ella constituye un punto de referencia para el desarrollo y continuación de este trabajo pues a pesar de que la venta de esta Biblioteca a Cuba no está permitida, la bibliografía disponible ofrece un conjunto de técnicas y algoritmos que potencian el trabajo con escenas dinámicas.
dPVS tiene un costo de $25000 por título.

## 1.7 Características de la “Scene Toolkit”

Uno de los objetivos de este trabajo es proponer el diseño de un módulo de clases que permita la manipulación eficiente de los objetos dinámicos en la Herramienta. Para lograr esto fue necesario el estudio de la arquitectura y funcionamiento de este software. En este epígrafe se exponen algunas de las características fundamentales de la Herramienta que constituyen puntos de partida para realizar la propuesta de diseño que se brindará en este trabajo.

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

Para lograr un *rendering* eficiente de la escena, primeramente se necesita tener el control de los objetos que en ella se encuentran. Existen dos variantes para organizar los objetos: [13]

- Variante 1: agrupar los objetos en una lista e iterar para escogerlos y aplicarles el *rendering*. Esto no es eficiente si cada objeto dibujable debe ser revisado.
- Variante 2: agrupar los objetos jerárquicamente de acuerdo a su localización espacial.

La segunda es obviamente más eficiente, y no es más que representar en un grafo toda la información manejable de la escena, tanto los objetos dibujables y no dibujables, como sus características. [13]

Generalmente, un grafo de escena presenta nodos intermedios (que contienen información para aplicar *render* a los objetos como posición, orientación y escala, volúmenes fronteras (utilizados para el *culling* jerárquico y la detección de intersecciones), estado de *render*…), y nodos hojas (objetos en cuestión).[13]

Por ejemplo, si se quisiera representar un auto en escena, algunas hojas del grafo se corresponderían con partes del auto como las ruedas, como objetos separados, y el auto sería un nodo intermedio que agruparía a los nodos hojas (sus partes). El nodo-grupo “auto” contendría información común a todas sus partes, y los estados que lo modifiquen modificarán a todas sus partes. [14]

De esta manera se organizan y almacenan los objetos de la escena en la Herramienta. El grafo de escena los contiene a todos y a través de la jerarquía establecida se facilita el trabajo de creación, actualización y destrucción de cada nodo (objeto).

### 1.7.2 Oclusión

La Herramienta cuenta con un módulo para la determinación de visibilidad a partir de regiones, que permite eliminar una considerable cantidad de

geometrías estáticas (80 % para cada celda) y aprovechando las particularidades de un entorno urbano, logra contribuir a que la simulación ocurra en tiempo real con independencia del tamaño de la escena.

La solución empleada, utiliza la idea de los planos soportadores [15], para determinar la visibilidad en una celda o región de la escena. Para cada región se calcula un conjunto de visibilidad conservativo a partir de la oclusión aportada por los objetos de la escena seleccionados como bloqueadores potenciales.
El algoritmo asume que la mayor parte de la geometría que realiza la oclusión en un escenario de ciudad, puede ser descrita a partir de prismas verticales. No se consideran como bloqueadores puentes u otros objetos no convertibles a prismas, aunque si pueden resultar ocultados durante el cálculo.[16]

Una vez concluida la estimación de la visibilidad, esta es guardada en un fichero, que contiene para cada región, la lista de objetos potencialmente visibles. Luego en tiempo de ejecución, el programa de representación de la escena en tiempo real, sólo tiene que determinar de los objetos visibles cuales están dentro del cono de visión. [16]

A menor tamaño de cuadrícula mayor es la oclusión (menos objetos se detectan como visibles) pero al mismo tiempo mayor es la cantidad de cuadrículas en las que se divide el mundo y crece el tamaño del archivo de almacenamiento.[16]

La solución utiliza *volúmenes contenedores* y *volúmenes bloqueadores* para los cálculos de oclusión. Los volúmenes contenedores encierran a toda la geometría y son utilizados para determinar si el objeto es visible o no. Generalmente se utilizan cajas contenedoras y esferas. [16]

Como volúmenes bloqueadores, se utilizaron prismas rectos que fueron colocados dentro de los edificios en la etapa de diseño. Un bloqueador dentro de un modelo 3D debe ser el mayor volumen contenido dentro de la figura.[16]

La Herramienta aplica la oclusión en un preprocesamiento y por ende sólo se tienen presente los objetos estáticos. Los objetos dinámicos no son tomados en cuenta, pero el conocimiento de los objetos estáticos potencialmente visibles puede proporcionar una base para aplicar la oclusión a los objetos dinámicos.

#### 1.7.2.1 Prueba de oclusión

Al asumir que la oclusión está dada fundamentalmente por prismas verticales, es posible realizar un análisis de la visibilidad en dos dimensiones, debiendo cumplirse los siguientes requisitos para que un objeto resulte ocultado: [16]

1- El objeto está entre los dos rayos que pasan por los extremos del bloqueador desde el punto de visión.
2- El objeto está detrás del bloqueador.
3- El ángulo de visibilidad ($\gamma$), del ocluido, tiene que ser menor que el del bloqueador. Este ángulo se forma entre el observador y la parte más alta de los objetos con respecto a la línea horizontal que contiene la altura del observador.

Estas condiciones se representan gráficamente en las Figuras 10 y 11.

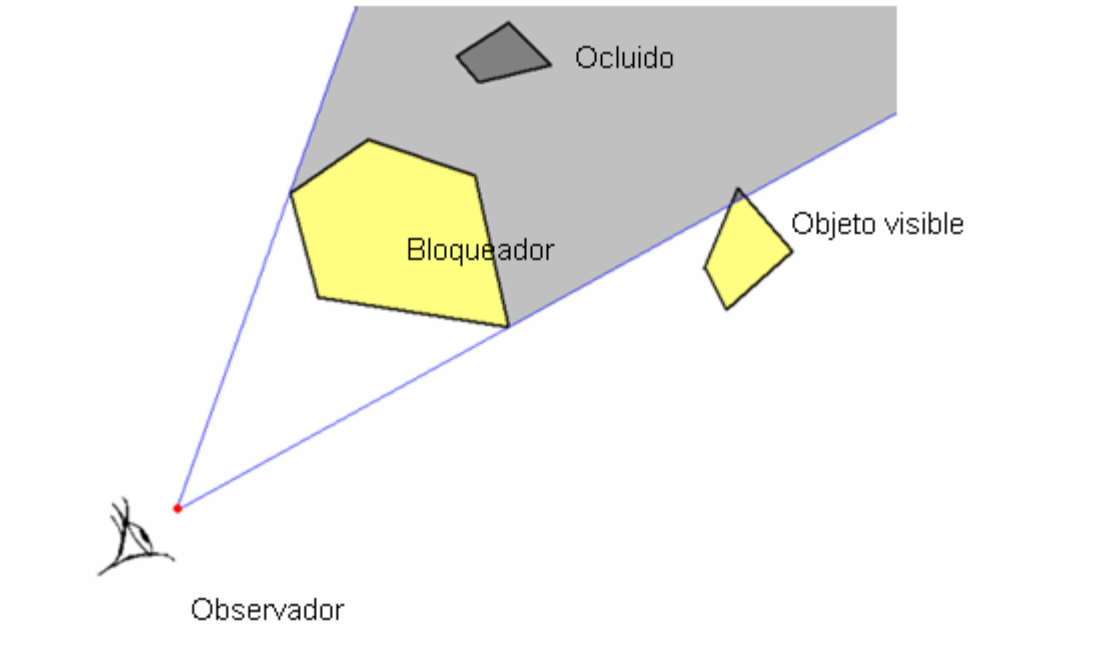


**Figura 10 Condiciones 1 y 2, el objeto se encuentra entre los rayos que pasan por el observador y los vértices extremos del bloqueador y se encuentra detrás del bloqueador.**

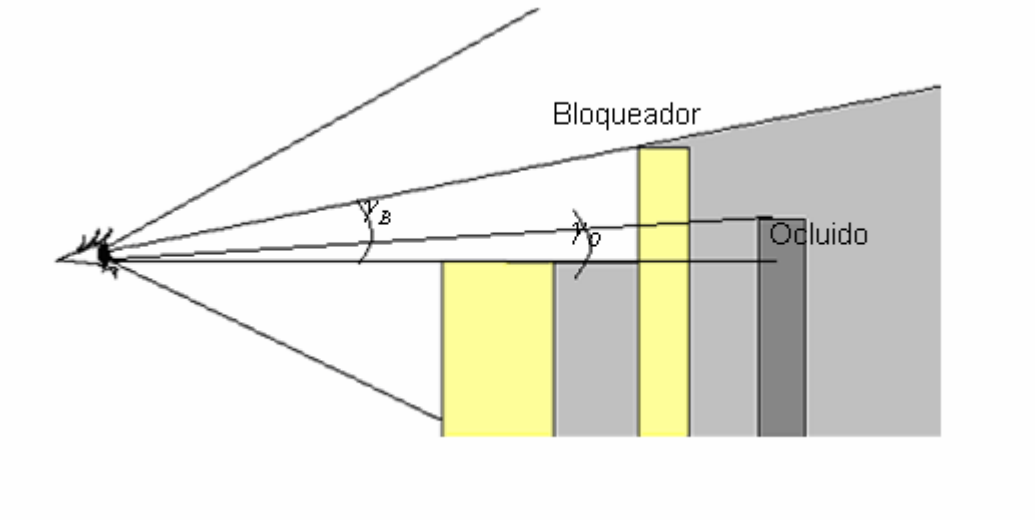


**Figura 11 Condición necesaria para el ocultamiento, el ángulo de visibilidad del ocluido $\gamma_O$, es menor que el ángulo de visibilidad del bloqueador $\gamma_B$.**

En la figura 14 se representa como se aplicó el principio de los planos soportadores. En este caso, como se está trabajando en el plano, se demuestra que al trazar líneas que soportan las siluetas del bloqueador (B) y la región (R), es posible obtener un área de oclusión de la visibilidad que es válida para toda la región R. El área de oclusión se encuentra sombreada en color gris. [16]

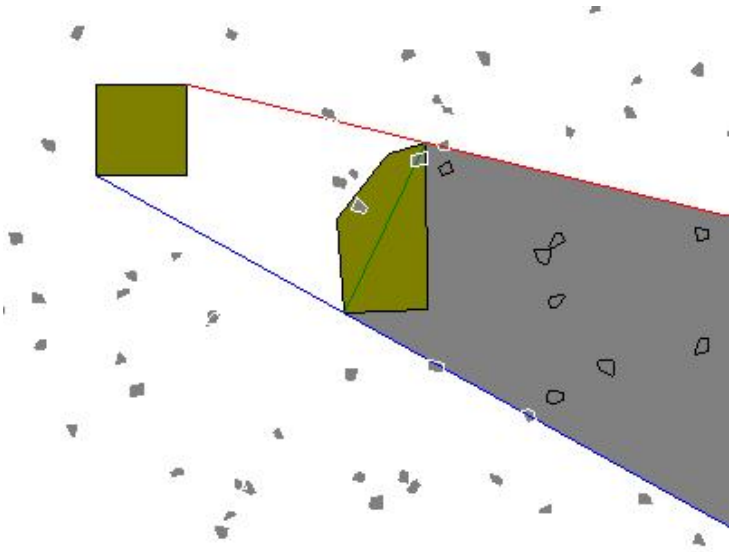

**Figura 12 Oclusión provocada por un bloqueador observando desde una región.**

Estas líneas de soporte se obtuvieron entre cada objeto bloqueador y la región de la siguiente forma:

1. Determinando la recta entre R y B, que tiene a todos los vértices de la región y el bloqueador a su derecha.
2. Determinando la recta entre R y B, que tiene a todos los vértices de la región y el bloqueador a su izquierda.[16]

## Conclusiones

Este capítulo sirve como base para la comprensión del tema que abordará este trabajo, se analizaron las principales características de los algoritmos, técnicas y tendencias actuales más empleadas en el manejo los objetos dinámicos en los Sistemas de Realidad Virtual. Mostrándose las deficiencias y ventajas que ofrece cada uno.

Se logró entender algunos aspectos (los necesarios para este trabajo) del funcionamiento de la *SceneToolkit.*

# Capítulo 2 Soluciones Técnicas

## Introducción

En este capítulo se proponen las soluciones técnicas para lograr una manipulación eficiente de los objetos dinámicos de las escenas virtuales y se explicará las ventajas que traerá consigo el uso de estas.

Se detallan los pasos del Algoritmo TBV y del Algoritmo de Reubicación de los objetos dinámicos así como la unión de ambos, llamado Algoritmo TBV+Reubicación.

## 2.1 Estructuras de datos utilizadas para manipular los objetos dinámicos.

Para el desarrollo de este proyecto se emplearán varias estructuras de datos con el objetivo de controlar y manipular los objetos dinámicos de la escena. Una estructura fundamental es la *Rejilla Regular* que se emplea para guardar la información espacial de los objetos dinámicos del entorno, y es la base para conocer el estado de visibilidad de cada objeto de la escena. El tamaño de las celdas, será grande, con el objetivo de agrupar una cantidad considerable de objetos dinámicos haciendo más eficiente su uso. Más adelante se abordará detalladamente el trabajo con esta estructura.

Se emplearán también los TBV (*Temporally Bounding Volumen)* basados en los "*bounding volume*" o "volumen frontera" de tipo caja, se eligió esta variedad ya que la mayoría de los objetos dinámicos que pueblan las escenas urbanas son geometrías irregulares por lo cual un *bounding volumen* de tipo caja es más eficiente para lograr un encierro ajustado al objeto y al espacio temporal que puede ocupar este.

Para el manejo de las posiciones de los objetos dinámicos en cada instante de su movimiento de utilizará un grafo que contendrá la información de las trayectorias del entorno 3D procedente del *fichero grf.*

Se utiliza además listas y una cola con prioridad para el manejo de los TBV.

## 2.2 Actualización de la Dinámica en la Rejilla Regular

Tanto para la cámara como para los demás objetos dinámicos la actualización de la Rejilla Regular es un aspecto fundamental.

A partir de la posición espacial del objeto a actualizar, se puede conocer en qué celda de la Rejilla se encuentra, de la siguiente manera:

i = PE(y – ymin/size).

j = PE(x – xmin/size).

donde :

i , j son las coordenadas (fila, columna) respectivamente de la celda en la Rejilla.

PE: parte entera

y : coordendas del objeto en el eje Y.

x : coordenadas del objeto en el eje X.

ymin : coordenadas del extremo min de la Rejilla en el eje Y .

xmin : coordendas del extremo min de la Rejilla en el eje X .

size : tamaño de las celdas.

## 2.3 Algoritmo Selección de Visibilidad en el Volumen de Visión (View Frustum Culling)

El *View Frustum* a utilizar por la cámara será de tipo 2D ya que la Rejilla Regular es 2D y además este proporciona una disminución considerable en los cálculos de visibilidad.

El recorrido del *View Frustum* se realizará en forma transversal de adentro hacia fuera como se surgieren en "*Dynamic Scene Occlusion Culling using a Regular Grill*" [4].

Para implementar este recorrido se parte de una línea de observación que siempre está dentro del *View Frustum* la cual se obtiene usando el algoritmo de línea de Bresenham's [12]. Para cada celda contenida en esta línea, llamadas celdas semillas, se realiza lo siguiente (Ver Figura 13 ) :

Sea (x, y) la posición de la celda semilla dado en coordenadas relativas, a la celda que contiene al punto de visión (*viewpoint*). Los recorridos serán:

(1) +y *y* –y Si |x| >|y|

(2) +x *y* –x Si |y| >|x|

(3) –x *y* –y Si ( |x| = |y|) y (x > 0) y (y > 0)

(4) +x *y* +y Si(|x| = |y| ) y (x<0) y ( y<0)

(5) +x *y* –y Si( |x| = |y| ) y (x<0) y (y>0)

(6) –x *y* +y Si(|x| = |y|) y (x>0) y (y<0).

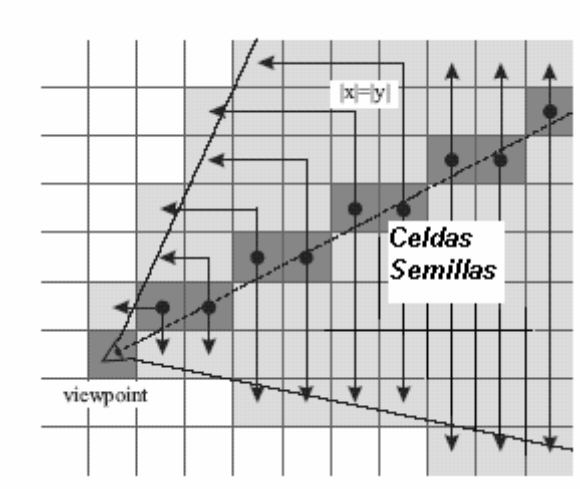


**Figura 13 Se muestra el V*iew-Frustum*, las celdas semilla (color gris oscuro) y las celdas visibles (color gris claro) resultado del recorrido.**

Como resultado de esto se detectan las celdas potencialmente visibles, para luego encuestar la visibilidad de los objetos contenidos en estas, que pueden ser de tipo:

- *Objetos Dinámicos*.
- TBV.

## 2.4 Algoritmo TBV para la manipulación de los objetos dinámicos. (A.TBV)

Para manipular los objetos dinámicos de la escena se empleará el algoritmo de los TBV que aparece en [6] pero con algunas modificaciones, por ejemplo: se suprime el Paso 4 y se modificó el Paso 2 eliminando el uso del *sello de tiempo* ya que por pruebas realizadas se detectó que para el caso particular de este trabajo resulta más costoso operacionalmente.

A continuación se muestran el algoritmo en pasos:

Estructuras de Datos empleadas

- *LV* Lista de objetos dinámicos visibles
- *QP-TBV* Cola con prioridad para los TBV
- *RG* -Estructura de Datos Espaciales Rejilla Regular.

**Aclaración**: En el primer *frame* todos los objetos son tomados como visibles.

Pasos a seguir para cada *frame*:

1. Corriendo el algoritmo de visibilidad si se encuentra una celda que contiene TBVs, estos TBVs son eliminados de *QP-TBV* y sus objetos son adicionados a *LV*. También sus nuevas posiciones son calculadas en dependencia del tiempo que fueron obviados.
2. Si cualquier TBV expira en la *QP-TBV*, este es removido de *QP-TBV* y se le asocia al objeto contenido otro TBV con un período de validación mayor. Este nuevo TBV es insertado en *QP-TBV*.
3. A todos los objetos de la lista de objetos dinámicos visibles que se encuentran dentro del View Frustum se les calcula su nueva posición y se actualiza en *RG*. El resto, pasan a ser invisibles y se le asocia un nuevo TBV que es insertado dentro de *QP-TBV* con su correspondiente período de validación.

Los períodos de validación de los TBV usan un algoritmo adaptativo para su selección:

- Si el TBV expira, de manera que el período fue corto, el próximo TBV se le asignará un período más largo.
- Si TBV se convierte en visible fue porque su TBV fue grande y/o el período de validación fue largo, por lo tanto el próximo TBV que se le asignará tendrá un período más corto.

## 2.5 Algoritmo de Reubicación de los Objetos de la Escena (A.Reubicación).

Para complementar el manejo eficiente de los objetos dinámicos de la escena se implementará un Algoritmo de Reubicación que tiene como objetivo fundamental disminuir el número de objetos dinámicos necesarios para simular interactividad en la escena. Una premisa importante para llevar a cabo esta etapa es conocer el estado de visibilidad de todos los objetos de la escena.

La idea principal es ubicar algunos de los objetos dinámicos no visibles de la escena en zonas potencialmente visibles de esta. Para esto se detectan las trayectorias del entorno que tienen la mayor probabilidad de ser visibles en los próximos frames y se reubica en ellas.

### 2.5.1 Criterio para iniciar la ejecución de Algoritmo de Reubicación

Al comenzar la ejecución del sistema, la cámara del entorno es ubicada en un nodo del grafo de trayectorias del entorno, este nodo se toma como el *nodo actual* por donde va la cámara, o sea, es el que marca el inicio de la trayectoria a seguir por la cámara, dicho nodo es actualizado siempre que la cámara arribe a un nuevo nodo del grafo de trayectorias del entorno.

A partir del *nodo actual* se obtienen las posibles trayectorias por la que se puede desplazar la cámara, esto es fácil conocerlo ya que cada nodo tiene la información de los nodos que están relacionados con él. Ver Figura 14

Cada una de esas posibles trayectorias por las que puede transitar la cámara en los próximos *frames* constituyen las zonas potencialmente visibles de la escena, los nodos que enlazan estas trayectorias (Links) con el *nodo actual* serán los *puntos de reubicación* y por tanto es allí donde se reubicarán los objetos.

La Reubicación de los objetos durante la visualización de la escena en el caso en que la cámara se encuentre en movimiento por la escena, se hace siempre que la cámara arribe a una nueva trayectoria, o sea, cuando el *nodo actual* cambie de valor. Para el caso en que la cámara pase un largo período de tiempo en reposo el nodo actual no cambiará de valor por tanto hay que variar el criterio para reubicar y en este caso se hará a partir de un rango de tiempo establecido, dicho rango se deja a elección del usuario ya que el definirá la frecuencia de reubicación en dependencia de las necesidades de la escena.

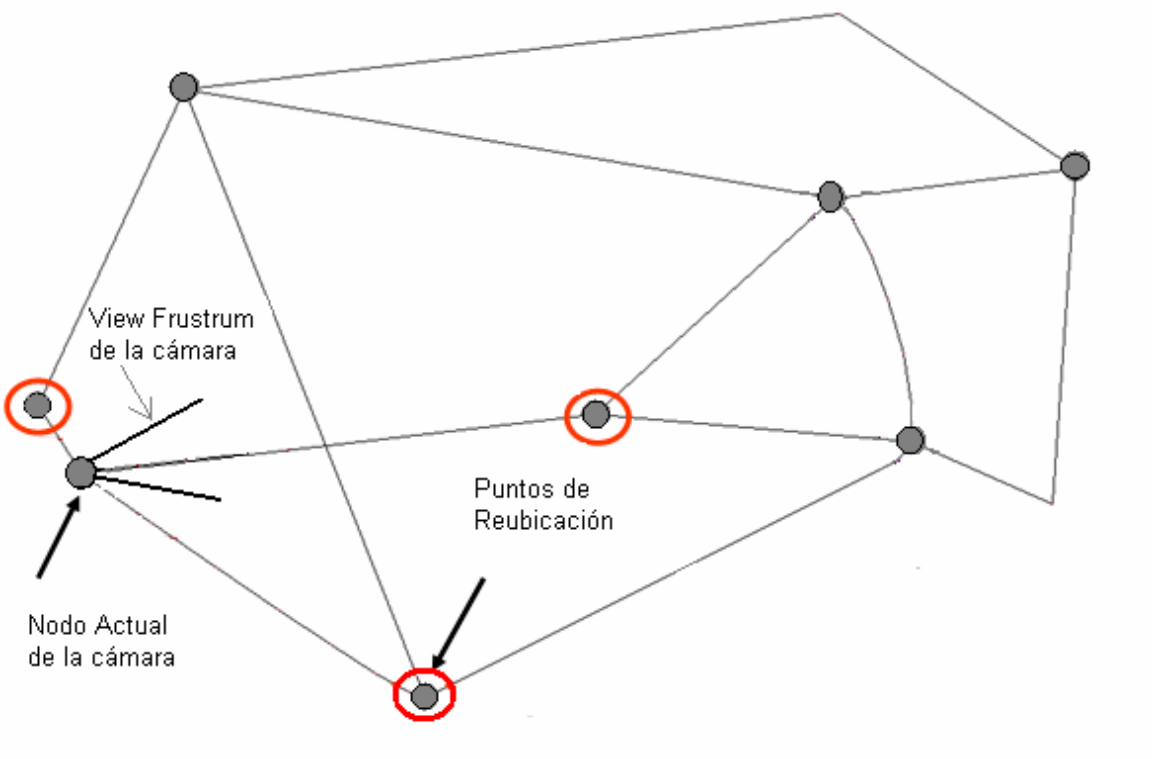


**Figura 14 Nodo actual de la cámara y las posibles trayectorias hacia donde se puede mover en los próximos *frames*.**

### 2.5.2 Aspectos a tener en cuenta en el Algoritmo

Teniendo en cuenta que el algoritmo es aplicable a entornos urbanos donde la cámara se puede mover en cualquier dirección sin restricciones, de manera impredecible, se hace indispensable chequear los elementos que se detallan en los párrafos siguientes con el objetivo de preservar el realismo en la escena simulada.

Como se explicaba en el epígrafe anterior, el algoritmo de Reubicación será ejecutado en el sistema siempre que la cámara arribe a un nuevo nodo del grafo de trayectorias del entorno. Imagine la visualización de una escena urbana en un sistema de simulación de conducción de autos, donde la cámara ha llegado a un nuevo nodo, en este instante se deben reubicar algunos de los objetos que no son visibles, supóngase que en el *frame* anterior un objeto dinámico fue

visible, por ejemplo un auto que pasó por la carretera donde se mueve la cámara. Ese objeto a pesar de que ya no es visible no puede ser reubicado porque puede suceder que la cámara gire, cambie la dirección de su movimiento bruscamente enfocando su *view frustum* en la trayectoria (carretera por donde se movía la cámara anteriormente) hacia donde se dirigía dicho objeto, si el objeto es reubicado en ese frame pues no aparecerá en esa trayectoria. Si esto ocurre, el usuario lo notará y evidentemente el realismo de la simulación se desvanecerá. Para impedir que esto suceda, el Algoritmo chequea que los objetos a reubicar, no estén ubicados en la *trayectoria anterior* por donde iba la cámara antes de efectuarse la última reubicación. Ver Figura 15

Se propone que este aspecto se chequee con más profundidad en posteriores versiones de este Algoritmo, ya que existen otros casos en los que la cámara hace un cambio brusco de dirección y puede que en las trayectorias que se hacen visibles en ese momento no aparezcan objetos que debían estar allí. En esta versión no se hace un análisis más detallado de estos puntos porque el *fichero grf* que se utiliza para obtener la información del entrono virtual no proporciona los datos necesarios para esto, por tanto, se recomienda que se estudie este tema para obtener una solución más completa y aumentar las potencialidades del algoritmo.

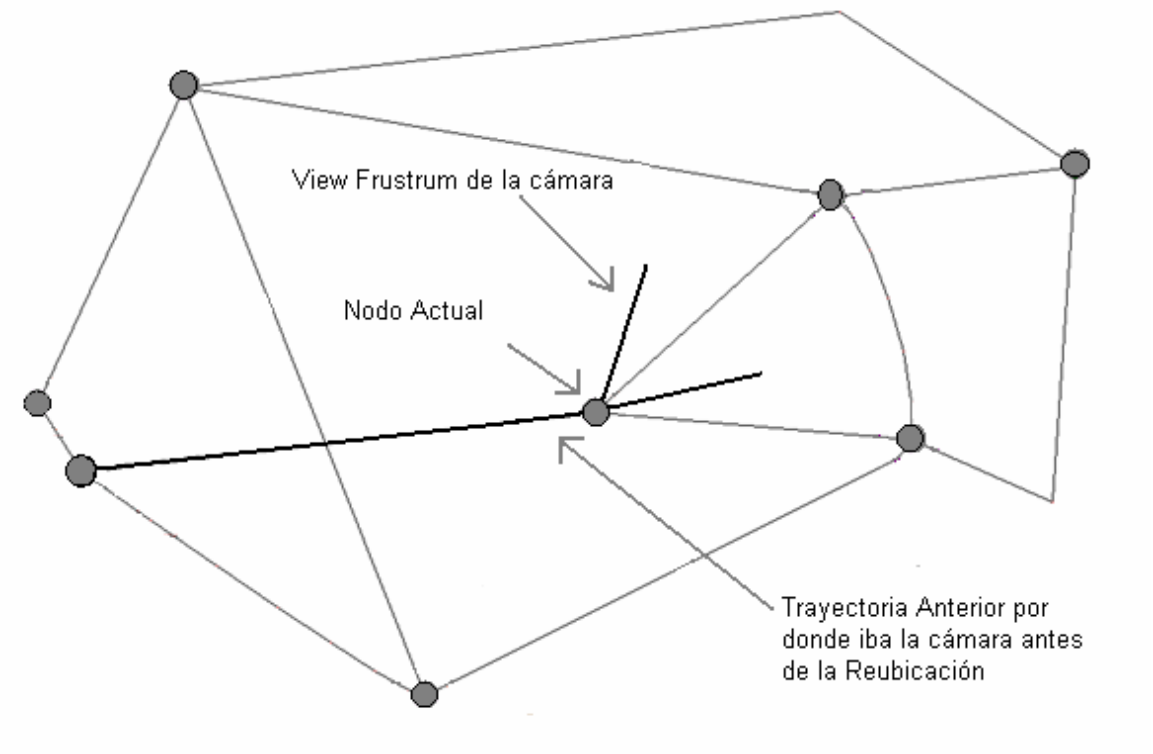


**Figura 15 Trayectoria Anterior por donde se desplazaba la cámara antes de la última Reubicación**

Hay otro detalle importante que tiene en cuenta el Algoritmo de Reubicación para supervisar el realismo de la escena simulada, se trata de la visibilidad de los nodos que constituyen los *puntos de reubicación*.

El estado "visible" de un nodo está determinado por la condición de que este esté ubicado dentro del *view frustum* de la cámara y que no sea ocluido por ningún objeto estático del entorno. Este último criterio de visibilidad que tiene en cuenta la oclusión, se expone como propuesta para que sea implementado en el "Módulo de Clases para la Manipulación de los Objetos Dinámicos en la SceneToolKit" del que va a formar parte este Algoritmo y se emplee para ello la manipulación de oclusión que tiene implementada la Herramienta (Epígrafe 1.7.2).

El hecho de que un nodo sea visible significa que las coordenadas que marcan la posición del nodo dentro del entorno virtual están dentro del cono de

visibilidad del usuario y si se reubica en ese punto, el objeto reubicado será visto por el usuario del sistema como ¨que cae del cielo¨, es decir, el objeto de pronto aparecerá en esa posición, lo cual le quita realismo a la escena.

De aquí que se hace necesario chequear la visibilidad de los nodos que constituyen puntos de reubicación, y si alguno de ellos es visible entonces hay que buscar un *nodo alternativo* donde Reubicar. Ver Figura 16.

Para obtener el *nodo alternativo* se determina cuales son los nodos que están relacionados con el nodo visible y arbitrariamente se elige uno de ellos.

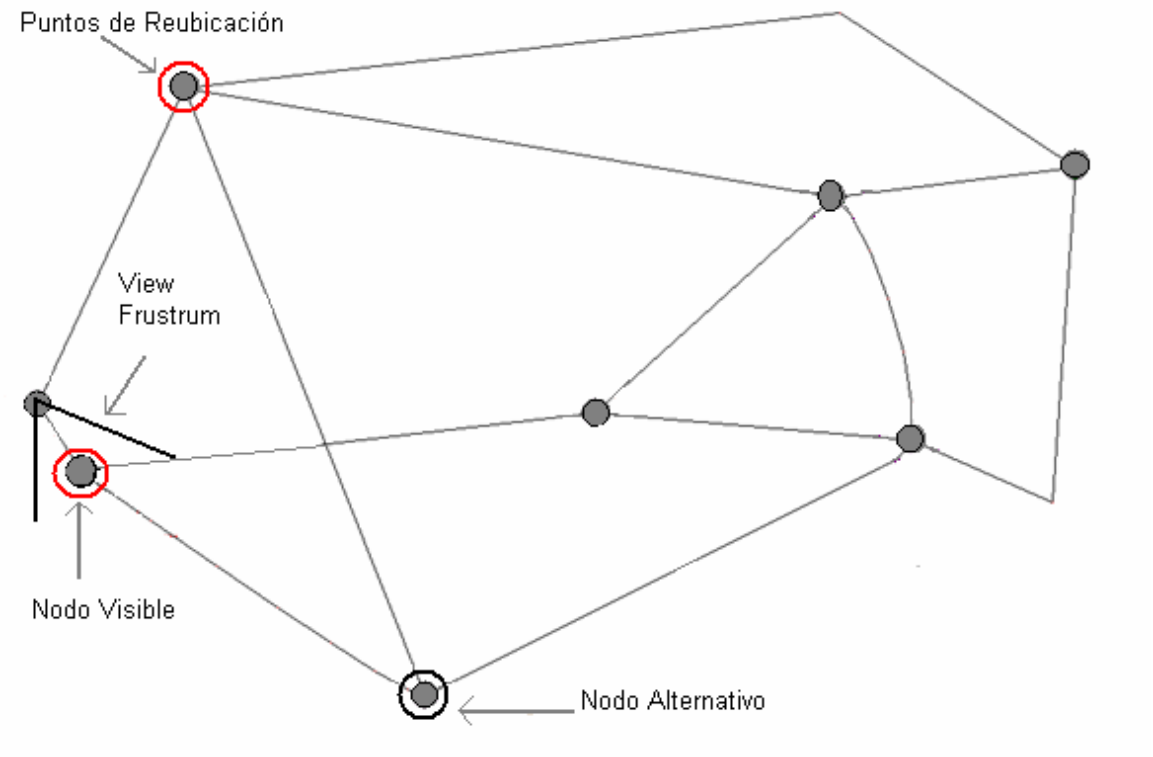


**Figura 16 Elección de un nodo alternativo en caso de que un punto de reubicación sea visible.**

### 2.5.3 Determinación de la cantidad de objetos que serán reubicados

Por cada punto de reubicación se reubicará cierto número de objetos. Esta *cantidad* se deja a decisión del usuario pues él establecerá el nivel de interactividad que primará en la escena, a esta *cantidad* se le adiciona la cantidad de objetos de la Lista de Objetos No Visibles (LNV) que están ubicados en la *trayectoria anterior* de la cámara (Epígrafe 2.2.5). A partir de esto se adapta la longitud de LNV para que cubra esta *cantidad,* o sea que, el número de objetos que hay en LNV debe ser suficiente para reubicar dicha *cantidad* de objetos por cada punto de reubicación.

Como se muestra en el pseudocódigo siguiente debe cumplirse que:

*CTA*: cantidad de objetos de LNV que están ubicados en la *trayectoria anterior* de la cámara.

*Cantidad* * Número de Puntos de Reubicación + *CTA* < Longitud de *LNV*

Como recomendación para este punto se propone que la cantidad de objetos a reubicar sea específica para cada nodo. La idea es que en la etapa de diseño del entorno virtual, los diseñadores especifiquen los nodos que están ubicados en zonas de la escena que requieran mucha actividad de objetos dinámicos, por ejemplo, una calle que bordee un supermercado, en esta zona el tráfico de objetos debe ser mayor que en otra, ya que sus características imponen mayor interactividad.

La diferenciación de los nodos en dependencia del lugar del entorno en que estén ubicados da una medida clara de la necesidad de dinamismo en cada parte de la escena, y hace que los resultados de la reubicación sean más eficientes.

### 2.5.4 Pasos del algoritmo de Reubicación

- *LNV*: lista de objetos dinámicos no visibles
- *LPR*: lista de *Puntos de Reubicación*.
- *COR*: cantidad de objetos a reubicar por cada nodo.
- *LOR*: lista de objetos a reubicar.
- *CNR*: cantidad de nodos que están relacionados con el nodo actual de la cámara.

1. Determinar el tamaño que va a tener *LNV* a partir de la cantidad de objetos que se van a reubicar por cada *punto de reubicación* (*COR*) definida por el usuario (Epígrafe 2.5.3).
2. Confeccionar *LPR*. Aquí se obtienen los nodos relacionados con el *nodo actual* de la cámara (que constituyen los posibles *puntos de reubicación*) y se verifica que no sean visible, en caso de que alguno lo sea, se busca un nodo alternativo y entonces este será el *punto de reubicación*. (Epígrafe 2.5.2)

   En el siguiente pseudocódigo se detallan estos aspectos:

```
Nodo_Actual ←Camara . Obtener_Nodo_Actual ();
Lista       Nodo_Actual . Obtener_Lista_Nodos_Relacionados ();

Desde i←1  hasta CNR  con paso 1
Inicio
  Si ( ! Lista[i]. Es_Visible ()), entonces:
    LPR. Adicionar (Lista[i]);
Sino,   entonces:
  Inicio
     Nodo← Lista[i]. Buscar_Nodo_Alternativo ();
     LPR. Adicionar (Nodo);
  Fin
Fin
```

3. Reubicar por cada nodo de LPR la *cantidad* seleccionada de objetos dinámicos.

   Como se puede apreciar en el pseudocódigo siguiente, en cada nodo de *LPR* se reubica la cantidad de objetos definida en el paso 1. Observe que siempre el objeto que se va a reubicar es el que está en la primera posición de LNV esto se debe a que esta lista es rotada cada vez que se reubica un objeto, o sea, el objeto que se reubica es el primero de LNV y una vez reubicado se elimina de LNV y se coloca al final de esta.

```
Para cada nodo n ∈ LPR, hacer:

Inicio
   Mientras (COR != 0), hacer
   Inicio
   Objeto_A_Reubicar = LNV [0];
    Si ( !Objeto_A_Reubicar. Esta_en_Trayectoria_Anterior ()), entonces:
    Inicio
      Objeto_A_Reubicar. Reubicar(n); //se reubica el objeto en el punto de reubicación n
       Decrementar COR;
    Fin
   Rotar LNV;
  Fin
Fin
```

Una recomendación válida para este Algoritmo es que una vez que se haya implementado el Módulo de Clases para la Manipulación Eficiente de los Objetos Dinámicos en la SceneToolkit, se tenga en cuenta que a la hora de reubicar los objetos del entorno 3D a estos se les varíe el modelo (cambios de textura, color), este detalle ofrece amplios beneficios para simular interactividad en la escena, permite que el usuario del sistema note variedad de objetos en el mundo virtual y aumente las potencialidades del Algoritmo.

## 2.6 Algoritmo General para la Manipulación Eficiente de los Objetos Dinámicos (A.TBV+Reubicación).

Los pasos del algoritmo general constituyen una mezcla los algoritmos expuestos anteriormente con algunas especificaciones.

Estructuras de Datos empleadas:

*QP-TBV* Cola con prioridad para los **TBV**

*LV* Lista de objetos dinámicos visibles

*LNV* Lista de objetos dinámicos no visibles

*LM* Lista de Modelos.

*RG* Estructura Datos Espaciales Rejilla Regular.

Aclaración: todos los objetos son considerados en el primer frame como no visibles.

**Pasos para cada** *frame*:

1. Ejecutar el algoritmo de Reubicación de los Objetos Dinámicos cuando se arriba a un nuevo nodo de *GTE* o a partir un rango de tiempo determinado, este último es en el caso en que la cámara se encuentre en reposo.

2. El trabajo con los TBV de los objetos reubicados será como sigue:
   - Si se reubicó en un nodo fuera del View Frustum el TBV será orientado en el sentido de la nueva trayectoria aumentando las potencialidades de ser visibles.
   - Si el nodo es ocluido el TBV será eliminado y el objeto contenido en este pasa a formar parte de la lista de *LV*.

3. Durante la ejecución del algoritmo de visibilidad (*View Frustum Culling )*si se encuentra una celda que contiene TBVs, estos son eliminados de *QP-TBV* y sus objetos son adicionados a *LV* y eliminados de *LNV*. También sus nuevas posiciones son calculadas en dependencia del tiempo que fueron obviados y actualizadas en *RG*.

4. Si cualquier TBV expira en *QP-TBV*, este es removido de la cola y se le asocia al objeto contenido otro TBV con un período de validación mayor. Este nuevo TBV es insertado en *QP-TBV*.

5. A todos los objetos de LV que se encuentran dentro del view frustum se les calcula su nueva posición y se actualiza en *RG*, el resto, son eliminados de *LV* e insertados en *LNV*

## Conclusiones

En este capítulo se han expuesto las soluciones técnicas para resolver el problema planteado en esta tesis. Se definieron los Algoritmos ha implementar: A.TBV, A.Reubicación y a partir de estos se conformó el A.TBV+Reubicación que mezcla las potencialidades de los dos algoritmos citados anteriormente y que será la solución a implementar en el módulo de manejo de objetos dinámicos de la herramienta.

Con esto quedan sentadas las bases técnicas para implementar los algoritmos y diseñar un Módulo de Clases para la Manipulación Eficiente de los Objetos Dinámicos en los entornos generados por la *SceneToolKit* lo cual constituye también un objetivo de esta tesis, y se abordará más adelante.

# Capítulo 3 Resultados Iniciales

## Introducción

En este capítulo se abordará algunos resultados obtenidos en pruebas iniciales hecha a los algoritmos, donde se realiza una proyección 2D de los resultados, por lo tanto, no se tiene en cuenta la carga gráfica que podría traer las proyecciones 3D, pero si deja ver las potencialidades de optimización de cálculo y de memoria a la hora de usar los algoritmos A.TBV y A.Reubicación indistintamente y en conjunto.

Las pruebas fueron hechas sobre un Demo que visualiza un grafo asociado a un entorno urbano por donde se mueve la cámara con su view frustum y se visualiza solo los objetos dinámicos que están dentro del mismo. El punto de comparación para las pruebas fue un Algoritmo Básico(A.Básico) que hace mover constantemente a los objetos dinámicos por todo el grafo y detecta los objetos que están dentro del view frustum haciéndolos visibles.

### 3.1 Resultados del A.TBV

En la Figura 17 se muestra las potencialidades del A.TBV. Este reduce sustancialmente el número de cálculos y muestra que al aumentar el número de objetos también aumenta la diferencia de los *frame* por segundos(fps) comparado con el A.Básico. Además la diferencia de los fps cuando aumenta la cantidad de objetos es pequeña lo que demuestra que es *sensible a la salida.*

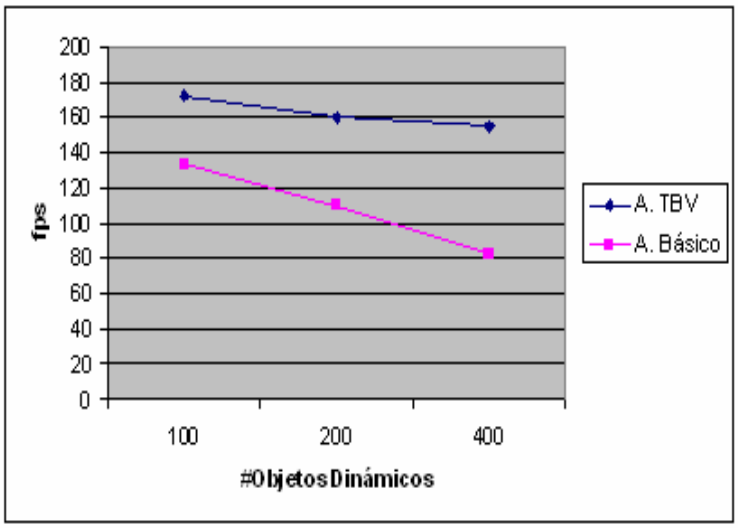


**Figura 17 Gráfica de los fps en función del número de objetos dinámicos.**

## 3.2 Resultados del A.Reubicación.

En esta prueba se comprobó cuantos objetos eran necesarios manipular por el A.Reubicación para lograr un promedio de objetos visibles similar al A.Básico. Como se muestra en la Tabla 1 se comprobó que con 20 objetos dinámicos y reubicando 3 cada vez que se cumple las condiciones de reubicación se logra aproximadamente el mismo promedio de visibilidad, para un 80% de reducción de los objetos dinámicos.

**Tabla 1 Tabla comparativa entre el A.Reubicación y A.Básico.**

| Algoritmos | Objetos dinámicos | Promedio de objetos visibles/frame |
|---|---|---|
| A.Reubicación | 20 | 4.9 |
| A.Básico | 100 | 5.4 |

## 3.3 Resultados del A.TBV+A.Reubicación.

En esta prueba se comprobó las potencialidades de la unión de estos dos algoritmos y como muestra la Figura 18 la reducción de los objetos y el aumento de los fps comparado con el A.Básico es sustancial. Con solo 90 objetos se puede lograr una fluidez aproximadamente igual que si hubiera 400 objetos moviéndose arbitrariamente, además los fps de los dos algoritmos experimentan una diferencia de 80 fps, siendo considerable la reducción de procesamiento de los objetos dinámicos. Además el ATBV+Reubicación es *sensible a la salida* e interactivo, cumpliendo con los requisitos que debe tener este tipo de algoritmos (Capítulo 1).

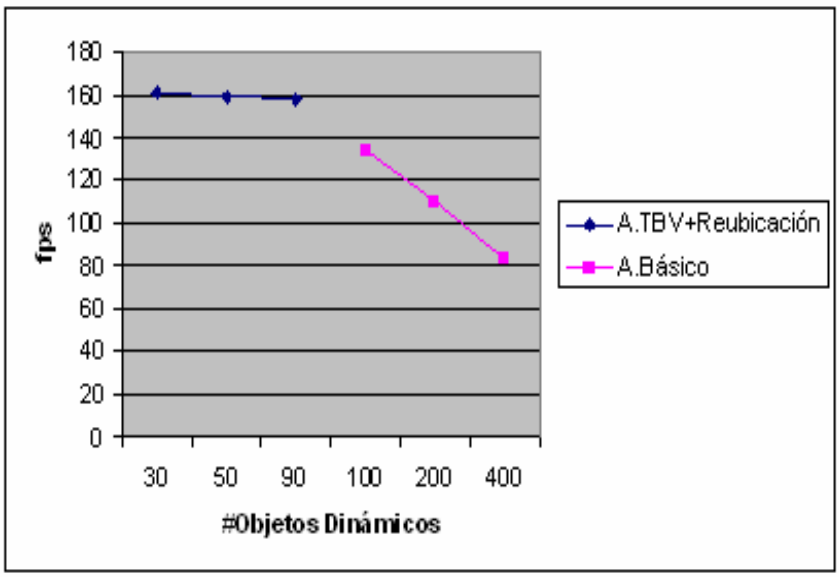

**Figura 18 Gráfica de los fps en función del número de objetos dinámicos.**

## Conclusiones

Como se puede ver estas pruebas demuestran la viabilidad de estos algoritmos para la manipulación de los objetos dinámicos en escenas urbanas. Se corrobora el cumplimiento de las exigencias necesarias que debe cumplir estos algoritmos como la *sensibilidad a la salida* así como el uso de los recursos de cómputo y memoria.

# Capítulo 4 Propuesta de Diseño del Módulo para la “SceeneTollkit”

## Introducción

En este capítulo se presentará el proceso de Diseño del Módulo de Clases para la Manipulación Eficiente de los Objetos Dinámicos que será acoplado a la herramienta *SceneToolKit. P*ara ello se toma como referencia al A.TBV+Reubicación. Esta propuesta se ajusta a la arquitectura actual de la herramienta e interactúa con varios de sus módulos ya implementados, permitiendo la reutilización y uso de sus potencialidades.

## 4.1 Reglas del Negocio

En este epígrafe se muestran las premisas que impone la incorporación del Algoritmo de TBV y el de Reubicación a la SceneToolKit:

Las celdas de la *Rejilla Regular* deben tener un tamaño relativamente grande para actualizar poco la estructura y disminuir los cálculos a la hora de detectar las celdas visibles en el v*iew frustum*. Además, la *rejilla regular* no puede eliminarse o modificarse hasta que no termine la ejecución de la aplicación.

El fichero que almacena la información espacial de las trayectorias del entorno debe ser de extensión (.grf). En caso de no existir el fichero en el directorio indicado, o ser de otro formato, se debe informar el error y no se ejecuta las funcionalidades correspondientes.

El tipo de *view frustum* debe ser 2D en caso contrario se informa sobre el error de uso y no se ejecutan las funcionalidades del módulo.

Se debe proporcionar los modelos necesarios para lograr una simulación realista a la hora de la reubicación de los objetos, en caso de no proporcionar dicha información se informa y de ser aceptado se ejecuta la funcionalidad del módulo sin realizar los cambios de modelos.

## 4.2 Modelo de Dominio

A continuación se muestra el diagrama de clases conceptuales perteneciente al Modelo de Dominio de este proyecto. Aquí se esquematiza el entorno que manejará el sistema a realizar y se relacionan los principales conceptos que forman parte de él. (Ver Figura 19)

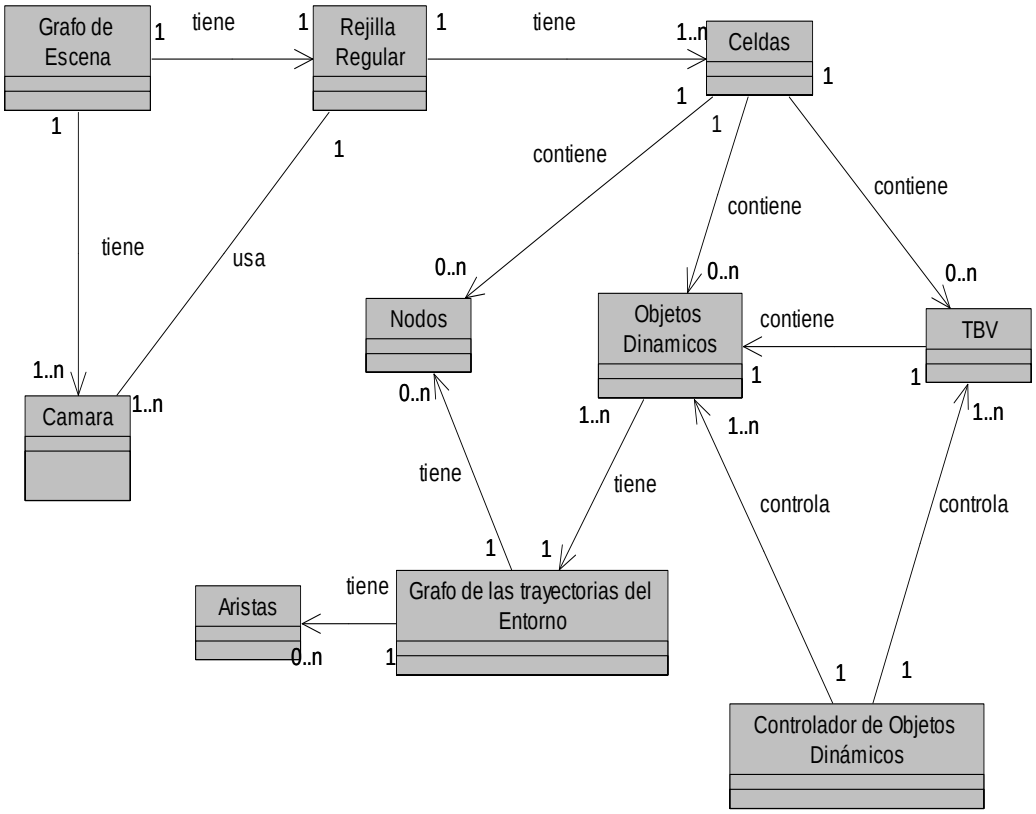


**Figura 19 Diagrama de clases conceptuales del Modelo de Dominio.**

### 4.2.1 Glosario de Términos del Dominio.

***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. (Este concepto no pertenece al dominio del presente trabajo sino al de la herramienta pero su definición es de suma importancia para comprender y modelar mejor el módulo)

***Celda***: espacio Regular en forma cuadrada en el cual se almacena la información de los objetos dinámicos y *TBV* contenidos en ella .

***Rejilla Regular:*** Estructura no jerárquica compuesta por celdas en forma matricial. (Epígrafe 1.1.2.1)

***Cámara***: elemento que permite la visualización en perspectiva del entorno.

***Grafo de las trayectorias del Entorno***: estructura que almacena la información sobre las trayectorias del entorno 3D por la cuales se pueden trasladar los objetos dinámicos, esta información proviene del *fichero grf.* Constituye un elemento fundamental para el control del movimiento de los objetos en el mundo virtual.

***Objetos Dinámicos***: objetos que se mueven por las trayectorias indicadas por el G*rafo de las trayectorias del Entorno.*

***TBV*** *(Temporal Bounding Volumen)*: volumen de encierro temporal ajustado a cualquier objeto dinámico, que enmarca el espacio que ocupa un objeto dinámico durante un tiempo determinado. (Epígrafe 1.3)

***Controlador de Objetos Dinámicos***: es el encargado de ejercer la manipulación eficiente de los objetos dinámicos, saber cuándo estos son visibles, cuándo se deben reubicar así como manejarlos según su TBV.

***Nodo***: elemento del grafo de trayectorias del entorno donde convergen aristas provenientes de otros nodos, constituye un punto de toma decisión dentro del grafo.

***Arista***: elemento de un grafo que conecta dos nodos, representa las trayectorias del entorno (Ejemplo: carreteras).

## 4.3 Captura de Requisitos

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

### 4.3.1 Requisitos Funcionales

1. Crear una Rejilla Regular
2. Configurar una Rejilla Regular.
3. Generar una Rejilla Regular.
4. Eliminar una Rejilla Regular.
5. Crear y Eliminar Celda.
6. Buscar todos los objetos de un tipo determinado ya sea Objetos Dinámicos o TBV que están en una celda determinada.
7. Insertar cualquier tipo de objeto en una celda.
8. Eliminar cualquier tipo de objeto de una celda.
9. Determinar en que celda o celdas está un Objeto Dinámico*.*
10. Determinar en que celda o celdas está un TBV.
11. Crear un TBV.
12. Eliminar un TBV.
13. Actualizar un TBV.
14. Determinar cuando un objeto es visible.
15. Determinar cuando un TBV es visible.
16. Determinar cuando una celda es visible en el *View Frustum.*
17. Informar cuando un objeto es visible.
18. Cargar un fichero grf.
19. Crear y Eliminar un Objeto Dinámico.
20. Actualizar la posición de los objetos dinámicos en la escena.
21. Asignar una trayectoria predeterminada a los objetos dinámicos

i) Seleccionar un número aleatorio entre una colección de números enteros.

22. Trasladar los objetos dinámicos por la escena.

i) Restar vectores 3d.
ii) Sumar Vectores 3d
iii) Multiplicar un vector 3d por un escalar.
iv) Multiplicar dos vectores 3d
v) Normalizar un vector 3d
vi) Calcular un vector ortogonal a un plano

23. Obtener los Link con los que esta relacionado cierto Nodo del *grafo de trayectorias del entorno*.
24. Obtener los Nodos que están relacionados con determinado Link del *grafo de trayectorias del entorno*.
25. Determinar nodo del grafo de trayectorias del entorno más cercano a la cámara.

i) Conocer en qué celda de la rejilla regular está ubicada la cámara.
ii) Calcular la distancia que media entre un nodo del grafo de trayectorias y la cámara.

26. Buscar los nodos que están en una celda determinada.
27. Determinar cuando un nodo del grafo de trayectorias del entorno es visible.

i) Determinar qué celda de la rejilla regular contiene a un nodo.
ii) Determinar si un nodo es visible.

28. Determinar los posibles caminos (aristas, nodos) que pueden ser accedidos por los objetos dinámicos a partir de un nodo dado.
29. Cambiar modelo a los objetos.
30. Actualizar delta_t para el movimiento de los objetos dinámicos.

### 4.3.2 Requisitos no Funcionales

- **Usabilidad**: los futuros usuarios del módulo serán programadores de la herramienta. El producto debe estar concebido para que el usuario piense qué desea hacer y no cómo hacerlo.

- **Rendimiento:** debe producir resultados *sensibles a la salida.*
- **Soporte:** debe estar disponible tanto para Window como Linux.
- **Legales:** se regirá por las normas ISO 9000.
- **Diseño e implementación**: se regirá por la filosofía de Programación Orientada a Objetos y el lenguaje a utilizar será el C++.

## 4.4 Modelo de casos de usos del sistema

En este epígrafe se reconocen los futuros actores del sistema a desarrollar y se definen, a partir de la agrupación de los requisitos funcionales anteriormente presentados, los casos de uso del sistema con sus respectivas expansiones. Además se muestra los diagramas de dependencias entre paquetes de casos de uso, así como el diagrama de casos de uso del sistema.

### 4.4.1 Actor del sistema

**Tabla 2 Actor del sistema**

| Actor | Justificación |
|---|---|
| Programador | Es el que podrá mandar a ejecutar las funcionalidades generales del módulo de clases y se beneficiará del resultado de las mismas. |
| | |

### 4.4.2 Casos de uso del sistema

**Tabla 3 Caso de Uso: Gestionar Celda**

| | |
|---|---|
| CU-GC | Gestionar Celda |
| **Actor** | Caso de Uso (CU-TOD) |
| **Descripción** | El caso de uso se inicia cuando el caso de uso que lo incluye solicita:<br>1. Buscar todos los objetos de un tipo determinado ya sea objetos dinámicos o **TBV** que estén en una celda.<br>2. Insertar o Eliminar un objeto de un tipo determinado en una celda. |
| **Referencia** | R-6,7,8 |

**Tabla 4 Caso de Uso: Gestionar Rejilla Regular**

| | |
|---|---|
| CU-GRR | Gestionar Rejilla Regular |
| **Actor** | Programador |
| **Descripción** | El Caso de Uso se inicia cuando el Programador desea:<br>1. Crear una Rejilla Regular.<br>2. Eliminar una Rejilla Regular. |
| **Referencia** | R-1,2,3,4,5 |

**Tabla 5 Caso de Uso: Determinar Visibilidad**

| | |
|---|---|
| CU-DV | Determinar Visibilidad |
| **Actor** | Caso de uso que lo incluye( CU-POD) |
| **Descripción** | El Caso de Uso se inicia cuando el caso de uso que lo incluye desea conocer que elementos son visibles, o sea cuales se encuentran dentro del View Frustum**.** Estos elementos pueden ser TBV u objetos dinámicos. En dependencia de si son visibles o no, se realiza un tratamiento diferenciado. En caso de que los objetos dinámicos no sean visibles se le asigna un TBV y pasa a formar parte de la lista de objetos no visibles. Por otra parte si un TBV es visible este deja de existir y su objeto pasa a formar parte de la lista de objetos visibles a los cuales se les verifica su visibilidad y se les actualiza su posición en dependencia del tiempo del TBV. |
| **Referencia** | R-11, 12, 13, 19, 20, 21, 30, CU-GC, CU-TOD |

**Tabla 6 Caso de Uso: Gestionar Expiración TBV.**

| CU-CETBV | Controlar Expiración TBV |
|---|---|
| **Actor** | Caso de uso que lo incluye(CU-POD) |
| **Descripción** | El Caso de Uso se inicia cuando el caso de uso que lo incluye solicita controlar el tiempo de expiración de los TBV. Luego de esto todos aquellos TBV que tengan un tiempo de vida mayor o igual a su máximo tiempo se les confirma que expiró, por lo tanto ese TBV deja de existir y se crea para el objeto contenido en este otro TBV con un tiempo de expiración mayor. |
| **Referencia** | R-11,12,13 |

**Tabla 7 Caso de Uso: Gestionar Objetos Dinámicos.**

| CU-GOD | Gestionar Objetos Dinámicos |
|---|---|
| **Actor** | Programador |
| **Descripción** | El caso de uso se inicia cuando el programador solicita:<br>1. Crear un objeto dinámico.<br>2. Eliminar un objeto dinámico. |
| **Referencia** | R-19 |

**Tabla 8 Caso de Uso: Cargar Grafo.**

| CU-COD | Cargar Grafo |
|---|---|
| **Actor** | Programador |
| **Descripción** | El caso de uso se inicia cuando el programador solicita que se cargue un fichero y entra la dirección donde se encuentra el mismo. El sistema verifica si la información referente al fichero entrada por el programador es válida, en caso afirmativo procede a cargar el fichero, de lo contrario se le informa al programador que hay errores en la información entrada. |
| **Referencia** | R-18 |

**Tabla 9 Caso de Uso: Reubicar Objetos Dinámicos.**

| | |
|---|---|
| CU-COD | Reubicar Objetos Dinámicos |
| **Actor** | Caso de uso que los incluye(CU-POD) |
| **Descripción** | El Caso de Uso se inicia cuando el caso de uso que lo incluye determina que ha llegado el momento de reubicar algunos de los objetos no visibles de la escena. En este instante:<br>• El sistema obtiene la información sobre el nodo del grafo más cercano a la cámara de la escena.<br>• El sistema busca los objetos dinámicos que no son visibles y los reubica en las zonas de la escena que tienen una alta probabilidad de ser visibles durante los próximos frames de visualización. |
| **Referencia** | R-13, 20, 21, 23, 24, 25, 26, 27, 28, 29, 30, CU-TOD |

**Tabla 10 Caso de Uso: Trasladar Objetos Dinámicos.**

| | |
|---|---|
| CU-TOD | Trasladar Objeto Dinámico |
| **Actor** | Caso de uso que los incluye(CU-DV) |
| **Descripción** | El Caso de Uso se inicia cuando el caso de uso que lo incluye solicita :<br>1. Asignar trayectoria un objeto dinámico para iniciar su movimiento.<br>2. Trasladar un objeto dinámico partiendo de una trayectoria predeterminada. |
| **Referencia** | R-20, 21, 22, CU-GC |

**Tabla 11 Caso de Uso: Procesar Objetos Dinámicos.**

| | |
|---|---|
| CU-COD | Procesar Objetos Dinámicos |
| **Actor** | Caso de Uso Hacer Ciclo( CU-HC) perteneciente al paquete Hacer Ciclo |
| **Descripción** | El caso de uso se inicia cuando el programador desea llevar a cabo el proceso de manipulación de los objetos dinámicos. En este caso, el sistema determina cuales objetos son visibles, actualiza sus posiciones y reubica los mismos para lograr un nivel de simulación aceptado. |
| **Referencia** | R-23,24,25,CU-DV, CU-ROD, CU-CETBV |

En el presente diagrama se presenta la dependencia existente entre el paquete de casos de uso *Hacer Ciclo* perteneciente a la Herramienta y el *Manipular Objetos Dinámicos* perteneciente al sistema del presente trabajo.

## 4.5 Relación entre paquetes de Caso de Uso

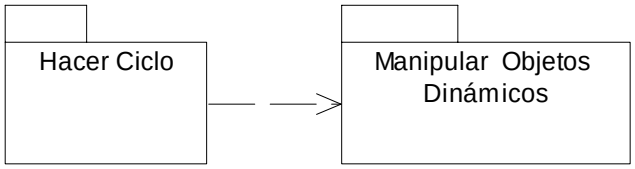


**Figura 20 Diagrama de Paquetes de Casos de Uso.**

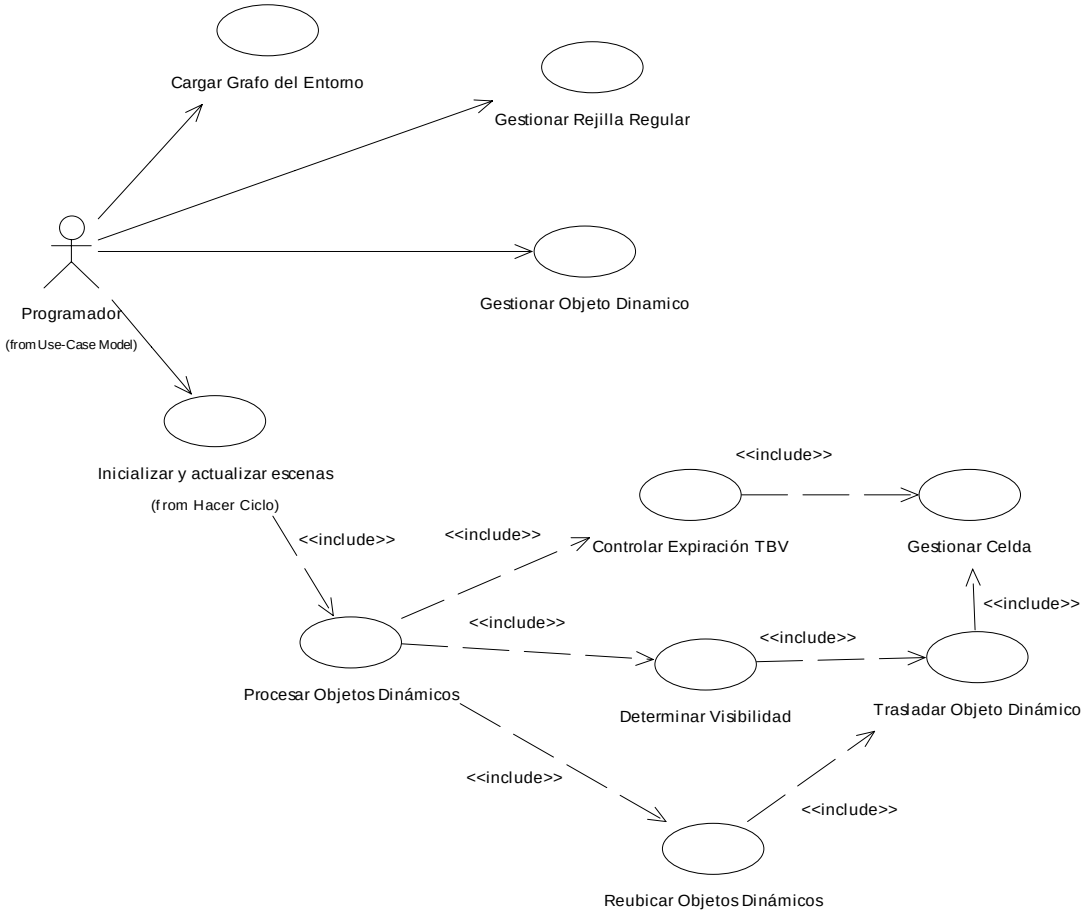


**Figura 21 Diagrama Casos de Uso del paquete Manipular Objetos Dinámicos.**

## 4.6 Expansión de los Casos de Uso

### Casos de uso expandidos

**Tabla 12 Expansión Caso de Uso Gestionar Celdas.**

| **Caso de uso** | |
|---|---|
| CU-GC | Gestionar Celda |
| **Propósito** | Realizar todas la operaciones sobre una celda. |
| **Actores :** Caso de Uso (CU-TOD) | |
| **Resumen:** El Caso de Uso se inicia cuando el caso de uso que lo incluye solicita:<br>1. Buscar todos los objetos de un tipo determinado ya sea objetos dinámicos o TBV que estén en una celda.<br>2. Insertar o Eliminar un objeto de un tipo determinado en una celda. | |
| **Referencias** | R-6,7,8 |
| **Precondiciones** | |
| **Flujo Normal de los Eventos** | |
| **Sección “Buscar Objetos”** | |
| **Acción del actor** | **Respuesta del sistema** |
| 1.Solicita obtener todos los objetos localizados en una celda según su tipo. | 1.1 Devuelve la lista de objetos del tipo especificado (TBV u objetos dinámicos). |
| **Sección “Insertar Objeto”** | |
| **Acción del actor** | **Respuesta del sistema** |
| 1. Solicita insertar un objeto localizados en una celda según su tipo. | 1.1 Inserta el objeto en su lista correspondiente según su tipo( TBV u objeto dinámicos). |
| **Sección “Eliminar Objeto”** | |
| **Acción del actor** | **Respuesta del sistema** |
| 1.Solicita eliminar un objeto localizado en una celda según su tipo. | 1.1 Elimina el objeto de su lista correspondiente, según su tipo ( TBV u objeto dinámico). |
| **Prioridad** | Crítico |

**Tabla 13 Expansión Caso de Uso Gestionar Rejilla Regular.**

| | |
|---|---|
| CU-GRR | Gestionar Rejilla Regular |
| **Propósito** | Realizar todas las operaciones sobre una Rejilla Regular. |
| **Actores :** Programador | |
| **Resumen:** El Caso de Uso se inicia cuando el Programador desea:<br>1. Crear una Rejilla Regular.<br>2. Eliminar una Rejilla Regular. | |
| **Referencias** | R-1,2,3,4,5 |
| **Precondiciones** | |
| **Flujo Normal de los Eventos** | |
| **Sección “Crear Rejilla Regular”** | |
| **Acción del actor** | **Respuesta del sistema** |
| 1. Solicita crear una Rejilla Regular con una cantidad específica de filas y columnas. | 1.1 Determina el tamaño que debe tener las celdas según el tamaño del entorno.<br>1.2 Crea un número de celdas igual a una cantidad de (filas x columnas). |
| **Sección “Eliminar Rejilla Regular”** | |
| **Acción del actor** | **Respuesta del sistema** |
| 1.Solicita eliminar una Rejilla Regular | 1.1 Elimina todas las celdas que la conforman así como la información contenida en estas. |
| Prioridad | Crítico |

**Tabla 14 Expansión Caso de Uso Determinar Visibilidad.**

| **Caso de uso** | | |
|---|---|---|
| CU-DV | Determinar Visibilidad | |
| **Propósito** | Determinar cuales objetos son visibles o lo que es lo mismo están dentro del *view frustum.* | |
| **Actores :** Caso de uso que lo incluye(CU-POD). | | |
| **Resumen:** El Caso de Uso se inicia cuando el caso de uso que lo incluye desea conocer que elementos son visibles, o sea cuales se encuentran dentro del *view frustum.* Estos elementos pueden ser TBV u objetos dinámicos. En dependencia de si son visibles o no, se realiza un tratamiento diferenciado. En caso de que los objetos dinámicos no sean visibles se le asigna un TBV y pasa a formar parte de la lista de objetos no visibles. Por otra parte si un TBV es visible este deja de existir y su objeto pasa a formar parte de la lista de objetos visibles a los cuales se les verifica su visibilidad y se les actualiza su posición en dependencia del tiempo del TBV. | | |
| **Referencias** | R-11,12,13,19,20,21,30, CU-GC, CU-TOD | |
| **Precondiciones** | Debe haberse creado un Rejilla Regular y los objetos a posicionados en las celdas que les corresponde según su posición. | |
| **Flujo Normal de los Eventos** | | |
| **Sección ""** | | |
| **Acción del actor** | | **Respuesta del sistema** |
| | | |
| 1. Solicita determinar cuales objetos son visibles. | | 1.1 Determina cuales celdas son visibles, o sea, tienen una parte o están completamente dentro del View Frustum.<br><br>1.2 Obtiene los TBV contenidos en cada celda visible (**invoca la Sección "Buscar Objetos" CU-GC**).<br><br>1.3 Determina cuales de los TBV son visible.<br><br>1.4 Elimina dichos TBV de sus celdas correspondientes.( **invoca la Sección "Eliminar Objeto" CU-GC**).<br><br>1.5 Actualiza el delta_t de los objetos, utilizando para esto el tiempo de vida de los TBVs que le corresponden. |

| | |
|---|---|
| | 1.6 Traslada los objetos según su delta_t.<br><br>1.7 Elimina los TBV visibles de la cola.<br><br>1.8 Actualizar el delta _t de los mismos con un valor constante.<br><br>1.9 Adiciona los objetos contenidos en el TBVs visibles a la lista de objetos visibles.<br><br>2.0 Traslada los objetos que se encuentran de la lista de objetos visibles por las trayectorias que le han sido asignada si se encuentran dentro del *view frustum*. ( **invoca a la Sección “Efectuar traslación del objeto” CU-TOD**). |
| **Flujo Alternativo** | |
| **Acción del Actor** | **Respuestas del sistema** |
| 2.0 Se les asigna un TBV, son eliminados de la lista de objetos visibles y adicionados a la lista de los no visibles. | |
| **Poscondiciones** | Son determinados todos los objetos dinámicos visibles que están listos para ser proyectados en la pantalla. |
| **Prioridad** | Crítico |

**Tabla 15 Expansión Caso de Uso Gestionar Expiración TBV.**

| **Caso de uso** | | |
|---|---|---|
| CU-CETBV | Controlar Expiración TBV | |
| **Propósito** | Controlar el proceso de expiración de los TBV | |
| **Actores :** Caso de uso que lo incluye(CU-POD). | | |
| **Resumen:** El Caso de Uso se inicia cuando el caso de uso de que lo incluye solicita controlar el tiempo de expiración de los TBV. Luego de esto todos aquellos TBV que tengan un tiempo de vida mayor o igual a su máximo tiempo se les confirma que expiró, por lo tanto ese TBV deja de existir y se crea para el objeto contenido en este otro TBV con un tiempo de expiración mayor. | | |
| **Referencias** | R-11,12,13,CU-GC | |
| **Precondiciones** | Todos los TBV deben estar almacenados en una cola con prioridad ordenada decrecientemente según su tiempo de vida. | |
| **Flujo Normal de los Eventos** | | |
| **Sección “”** | | |
| **Acción del actor** | | **Respuesta del sistema** |
| | | |
| 1. Solicita actualizar los TBV. | | 1.1 Determina si el primer TBV de la cola con prioridad alcanzó su tiempo máximo de vida o si es igual a 0, que indica que se hizo visible.<br><br>1.2 Elimina el TBV de la cola con prioridad.<br><br>1.3 Elimina el TBV de la lista de objetos perteneciente a las celda (s) que lo contiene( **invoca a la Sección ”Eliminar Objetos” CU-GC**)<br><br>1.4 Elimina el TBV.<br><br>1.5 Se crea un TBV con un tiempo de expiración mayor al que tenía el TBV anteriormente.<br><br>1.6 Adiciona dicho TBV a la cola con prioridad. |

| | |
|---|---|
| **Flujo Alternativo** | |
| **Acción del Actor** | **Respuestas del sistema** |
| 1.1 En caso de no expirar el primer TBV ni ser cero su tiempo de expiración solo se actualiza su tiempo de vida y | el caso de uso finaliza. |
| **Poscondiciones** | Es actualizada la cola de los TBV. |
| **Prioridad** | Crítico |

**Tabla 16 Expansión Caso de Uso Gestionar Objetos Dinámicos.**

| **Caso de uso** | |
|---|---|
| CU-GOD | Gestionar Objeto Dinámico. |
| **Propósito** | Crear, Eliminar. |
| **Actores :** Programador | |
| **Resumen:** El caso de uso se inicia cuando el programador solicita:<br>2. Crear un objeto dinámico.<br>3. Eliminar un objeto dinámico. | |
| **Referencias** | |
| **Precondiciones** | Debe haberse ejecutado el caso de uso Cargar Grafo. |
| **Flujo Normal de los Eventos** | |
| **Sección “Crear Objeto Dinámico”** | |
| **Acción del actor** | **Respuesta del sistema** |
| 1. Solicita crear un objeto dinámico con un grafo asociado. | 1.1 Crea un objeto dinámico y le asocia el grafo. |
| **Sección “Eliminar Objeto Dinámico”** | |
| **Acción del actor** | **Respuesta del sistema** |
| 1.Solicita eliminar un objeto dinámico. | 1.1 Elimina el objeto dinámico. |
| **Poscondiciones** | Se crea, elimina y actualiza un objeto dinámico. |
| **Prioridad** | Crítico |

**Tabla 17 Expansión Caso de Uso Cargar Grafo.**

| **Caso de uso** | |
|---|---|
| CU-CG | Cargar Grafo |
| **Propósito** | Cargar un fichero grf |
| **Actores :** Programador | |
| **Resumen:** El caso de uso se inicia cuando el programador solicita que se cargue un fichero grf y entra la dirección donde se encuentra el mismo. El sistema verifica si la información entrada por el programador es válida, en caso afirmativo procede a cargar el fichero, de lo contrario se le informa al programador que hay errores en la información . | |
| **Referencias** | R-18 |

| **Acción del actor** | **Respuesta del sistema** |
|---|---|
| 1. Solicita cargar un fichero grf indicando el nombre del fichero y la dirección donde se encuentra. | 1.1 Verifica que la información entrada por el programador sea válida.<br>1.2 Se abre el fichero.<br>1.3 Se lee y almacena la cantidad de polilíneas que se especifican en el fichero.<br>1.4 Se lee y almacena la cantidad de nodos que se especifican en el fichero.<br>1.5 Se lee y almacena el resto de la información contenida en el fichero sobre las polilíneas, los nodos y los links.<br>1.6 Se crea un grafo para manipular y almacenar la información obtenida del fichero.<br>1.7 Se cierra el fichero |
| **Flujo alternativo** | |
| **Acción del actor** | **Respuesta del sistema** |
| 1.2. El sistema muestra un mensaje al programador informando que la información referente al fichero no es válida. | |
| **Poscondiciones** | El grafo de trayectorias de la escena almacena toda la información que contiene el fichero .grf |
| **Prioridad** | Crítico |

**Tabla 18 Expansión Caso de Uso Trasladar Objetos Dinámicos.**

| **Caso de uso** | | |
|---|---|---|
| CU-TOD | Trasladar Objeto Dinámico | |
| **Propósito** | Asignar al objeto dinámico la trayectoria por la que va a iniciar su movimiento. Efectuar la traslación del objeto. | |
| **Actores :** Caso de uso que lo incluye(CU-DV) | | |
| **Resumen:** El Caso de Uso se inicia cuando el caso de uso que lo incluye solicita :<br>2. Asignar trayectoria un objeto dinámico para iniciar su movimiento.<br>3. Trasladar un objeto dinámico partiendo de una trayectoria predeterminada. | | |
| **Referencias** | R -20,21,22,23,24,CU-GC | |
| **Precondiciones** | Los objetos dinámicos deben estar creados.<br>Deber haberse ejecutado el CU-CG. | |
| **Flujo Normal de los Eventos** | | |
| **Sección “** Asignar trayectoria un objeto dinámico **”** | | |
| **Acción del actor** | **Respuesta del sistema** | |
| | 1.1 Actualiza el Nodo_Inicial de la trayectoria del objeto dinámico.<br><br>1.2 Actualiza el Nodo_Destino_Inmediato de la trayectoria del objeto dinámico.<br><br>1.3 Determina el Link que une al Nodo_Inicial y al Nodo_Destino_Inmediato (Link_Actual).<br><br>1.4 Determina cualquier Link del grafo de trayectorias del entorno que esté enlazado con el Nodo_Inicial, excepto el Link_Actual (Link_Anterior).<br><br>1.5 Determina el nodo que contiene al Link_Anterior y que es diferente del Nodo_Inicial (Nodo_Anterior).<br><br>1.6 Elige cualquier Link del Nodo_Destino_Inmediato, excepto el Link_Actual (Link_Siguiente).<br><br>1.7 Inicializa Los parámetros de la trayectoria inicial del movimiento con los datos determinados | |

| | |
|---|---|
| | anteriormente. |
| **Sección “**Efectuar traslación del objeto**”** | |
| **Acción del actor** | **Respuesta del sistema** |
| | 1.1 Determina el vértice de la polilínena (Link_Actual) donde se encuentra el objeto, este sería uno de los vectores 3d que componen la polilínea, es nombrado Vértice_Actual.<br><br>1.2 Determina el vértice de la polilínea hacia donde se dirige el objeto, o sea el vector que le sigue al Vértice_Actual en la colección de vectores 3d que contiene la polilínea.<br><br>1.3 Obtiene la posición vectorial del objeto teniendo en cuenta el desplazamiento lateral que tiene con respecto a la línea centro de la trayectoria (Se hace uso las *propiedades del vector ortogonal*).<br><br>1.4 Calcula la nueva posición del objeto, teniendo en cuenta el desplazamiento que tiene asignado.<br><br>1.5 Asigna al objeto la nueva posición antes calculada.<br><br>1.6 Elimina al objeto de la celda(s) donde se encontraba anteriormente ( **Invoca a la Sección “Eliminar Objeto” del CU-GC**).<br><br>1.7 Inserta al objeto en la celda (s) que se encuentra actualmente. (**Invoca a la Sección “Insertar Objeto” del CU-GC**). |
| **Poscondiciones** | El objeto se traslada por la escena. |
| **Prioridad** | Crítico |

Se recomienda que para comprender perfectamente cada aspecto de la siguiente descripción se remitan al Epígrafe 2.5.2, dónde se definen algunos términos importantes para este caso de uso.

**Tabla 19 Expansión Caso de Uso Reubicar Objetos Dinámicos.**

<table>
<tr><th colspan="3">Caso de uso</th></tr>
<tr><td>CU-ROD</td><td colspan="2">Reubicar Objetos Dinámicos</td></tr>
<tr><td>Propósito</td><td colspan="2">Llevar a cabo el proceso de reubicación de los objetos dinámicos, que consiste en :<ul><li>Determinar los objetos a reubicar chequeando que estos no estén localizados en la trayectoria anterior de la cámara.</li><li>Determinar el nodo actual por donde va la cámara.</li><li>Determinar la zona potencialmente visible de la escena (puntos de reubicación).</li><li>Chequear la visibilidad de los objetos a reubicar.</li><li>Colocar estos objetos en los puntos de reubicación.</li></ul></td></tr>
<tr><td colspan="3">Actores : Caso de uso que lo incluye (CU-POD)</td></tr>
<tr><td colspan="3">Resumen: El Caso de Uso se inicia cuando el caso de uso que lo incluye determina que ha llegado el momento de reubicar algunos de los objetos no visibles de la escena. En este instante:<ul><li>El sistema obtiene la información sobre el nodo del grafo más cercano a la cámara de la escena.</li><li>El sistema busca los objetos dinámicos que no son visibles y los reubica en las zonas de la escena que tienen una alta probabilidad de ser visibles durante los próximos frames de visualización.</li></ul></td></tr>
<tr><td>Referencias</td><td colspan="2">R-13,20,21,23,24,25,26,27,28,29,30,CU-TOD</td></tr>
<tr><td>Precondiciones</td><td colspan="2">Los objetos dinámicos de la escena deben tener actualizada la información referente a su estado de visibilidad.</td></tr>
<tr><th colspan="3">Flujo Normal de los Eventos</th></tr>
<tr><th colspan="3">Sección “”</th></tr>
<tr><th>Acción del actor</th><th colspan="2">Respuesta del sistema</th></tr>
<tr><td>1 Determina que ha llegado el momento de Reubicar los objetos dinámicos que no son visibles.</td><td colspan="2">1.1 Determina de los objetos dinámicos que no están siendo visibles aquellos que se van a reubicar. Para esto se chequea que los objetos no estén ubicados en la trayectoria anterior a la cámara.<br>1.2 Obtiene información sobre cuál es el nodo del grafo de trayectorias del entorno, dónde está ubicada la cámara en la escena (nodo actual).<br><br>1.3 Determina los nodos hacia los cuales se puede llegar a partir del nodo actual. Estos marcan las posibles trayectorias por donde se puede desplazar la cámara en</td></tr>
</table>

| | |
|---|---|
| | los próximos *frames* (puntos de reubicación)<br><br>1.4 Chequea la visibilidad de los puntos de reubicación. Los nodos que sean visibles serán sustituidos por un nodo alternativo, que se elige de la lista que contiene los nodos que están relacionados con el visible.<br><br>1.5 Asigna a los objetos dinámicos a reubicar las trayectorias de traslación partiendo de los *puntos de reubicación* seleccionados anteriormente. **(invoca a sección “ Asignar trayectoria a un objeto dinámico ” CU-TOD**).<br><br>1.6 Cambia el modelo geométrico al objeto reubicado.<br><br>1.7 Actualiza el TBV que lo contiene orientándolo en el sentido de la nueva trayectoria a seguir. |
| **Poscondiciones** | Los Objetos no visibles seleccionados son reubicados a zonas potencialmente visibles de la escena. |
| **Prioridad** | Crítico |

**Tabla 20 Expansión Caso de Uso Procesar Objetos Dinámicos.**

| **Caso de uso** | |
|---|---|
| CU-POD | Procesar Objetos Dinámicos. |
| **Propósito** | Controlar el proceso de manipulación de objetos dinámicos |
| **Actores :** Caso de uso que lo incluye( Caso de uso **Inicializar y Actualizar Escena** perteneciente al paquete Hacer Ciclo) | |
| **Resumen:** El caso de uso se inicia cuando el programador desea llevar a cabo el proceso de manipulación de los objetos dinámicos. En este caso, el sistema determina cuales objetos son visibles, actualiza sus posiciones y reubica los mismos para lograr un nivel de simulación aceptada. | |
| **Referencias** | R-23,24,25,CU-DV, CU-ROD, CU-CETBV |
| **Precondiciones** | Debe haberse creado una Rejilla Regular y los objetos dinámicos a manipular .También deben estar posicionados en las celdas que les corresponde según su posición. Además debe ejecutarse el caso de uso Cargar Grafo. |
| **Flujo Normal de los Eventos** | |
| **Sección “”** | |
| **Acción del actor** | **Respuesta del sistema** |
| | |
| 1. Solicita llevar a cabo el proceso de manipulación de los objetos dinámicos. | 1.1 Determina cuando la cámara llega un nuevo nodo del grafo de trayectoria del entorno y actualizar esta información o si lleva un tiempo X en reposo definido con anterioridad.( **invoca al caso de uso CU-ROB**).<br><br>1.2 Actualiza la información sobre el estado de visibilidad de los objetos dinámicos de la escena(**invoca al caso de uso CU-DV**).<br><br>1.3 Determina si alguno(s) de los TBV expiró(**invocar CU-CETBV**) |
| **Poscondiciones** | Se visualizan los objetos dinámicos visibles y se Reubican aquellos no visibles. |
| **Prioridad** | Crítico |

## 4.7 Diseño del Sistema

## Introducción

En este capítulo se muestran los artefactos fundamentales que describen el diseño del módulo de clases. Mediante los diagramas de paquetes se demuestran las fuertes dependencias que tiene este módulo con los paquetes de la herramienta, debido a que todo este diseño se basa en la arquitectura de la misma.

Se muestran los diagramas de secuencias asociados a los casos de uso expuestos en el capítulo anterior así como los diagramas de clases asociados a los subpaquetes de diseño del módulo.

### 4.7.1 Diagrama de Paquetes de Diseño

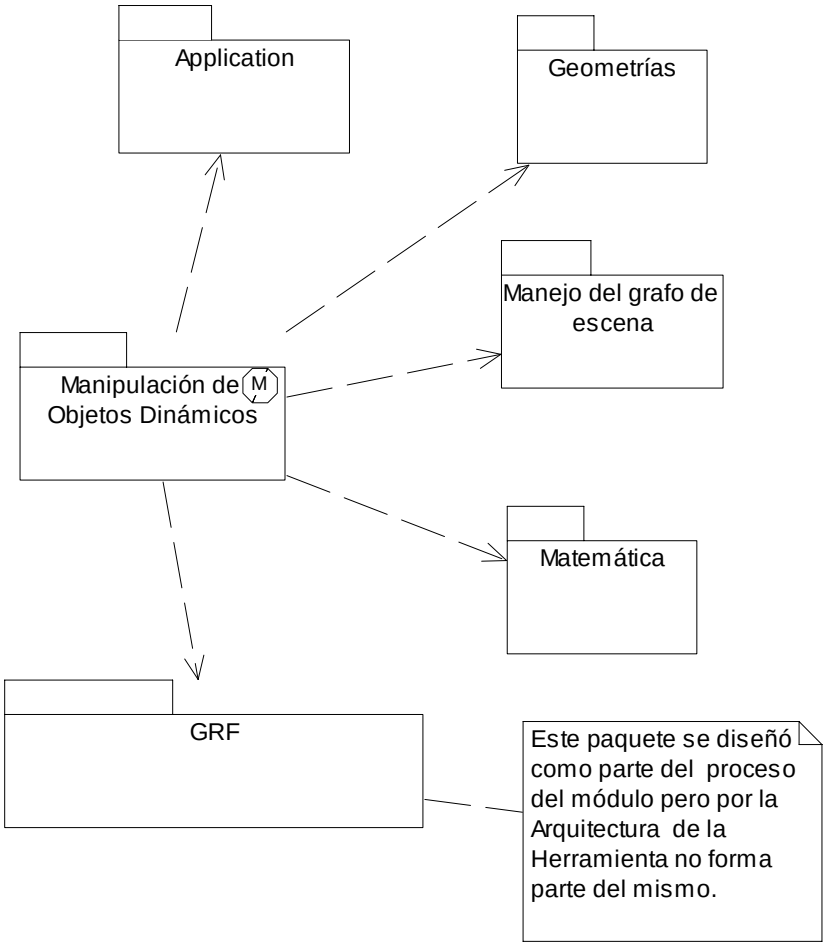


**Figura 22 Diagrama dependencia del Módulo “Manipulación de Objetos Dinámicos” con otros módulos de la herramienta.**

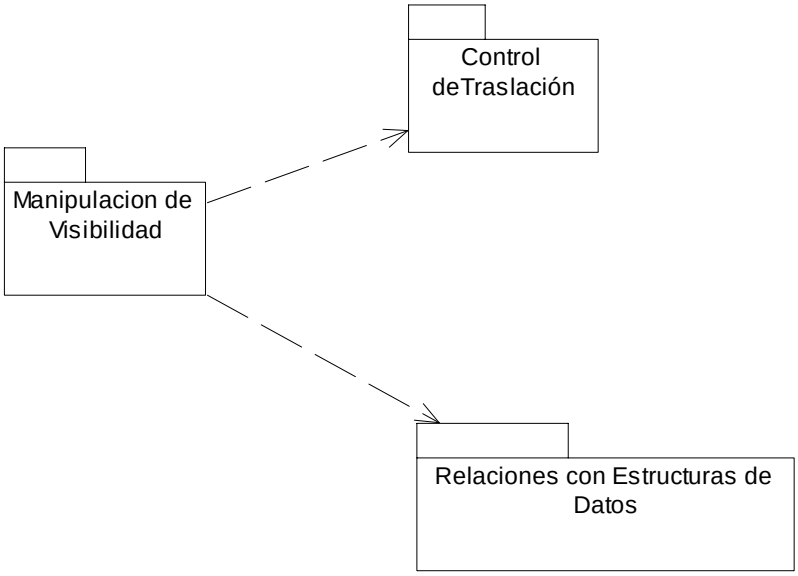


**Figura 23 Diagrama de SubPaquetes del Módulo.**

## 4.7.2 Diagrama de Clases de Diseño

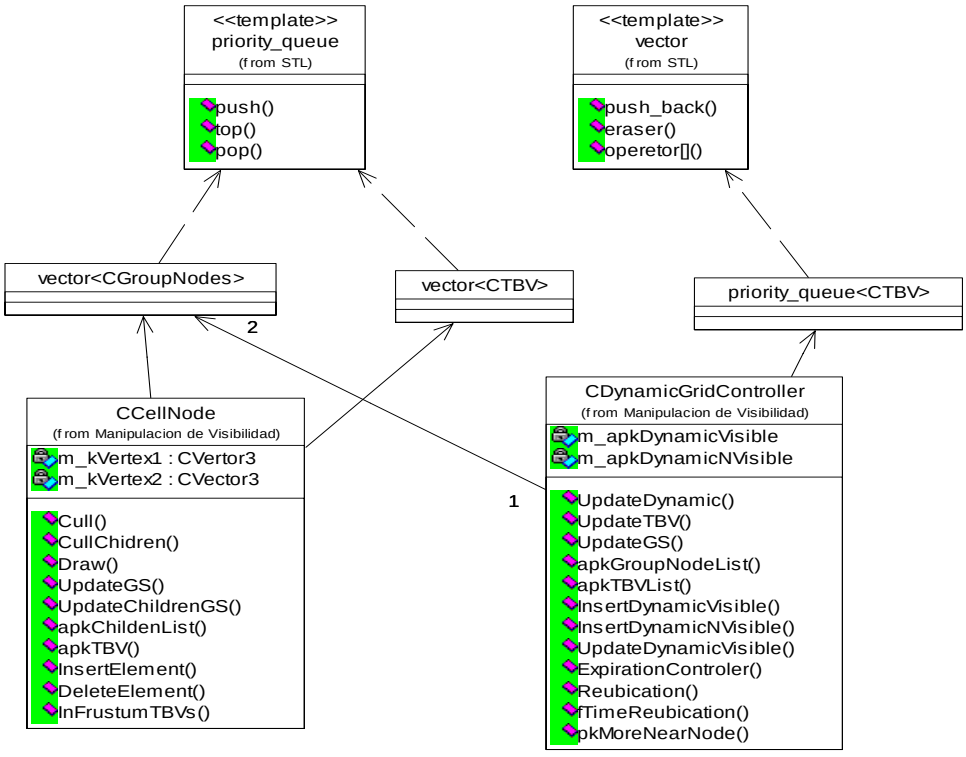


**Figura 24 Diagrama de Clase Diseños “Relaciones con Estructuras de Datos”.**

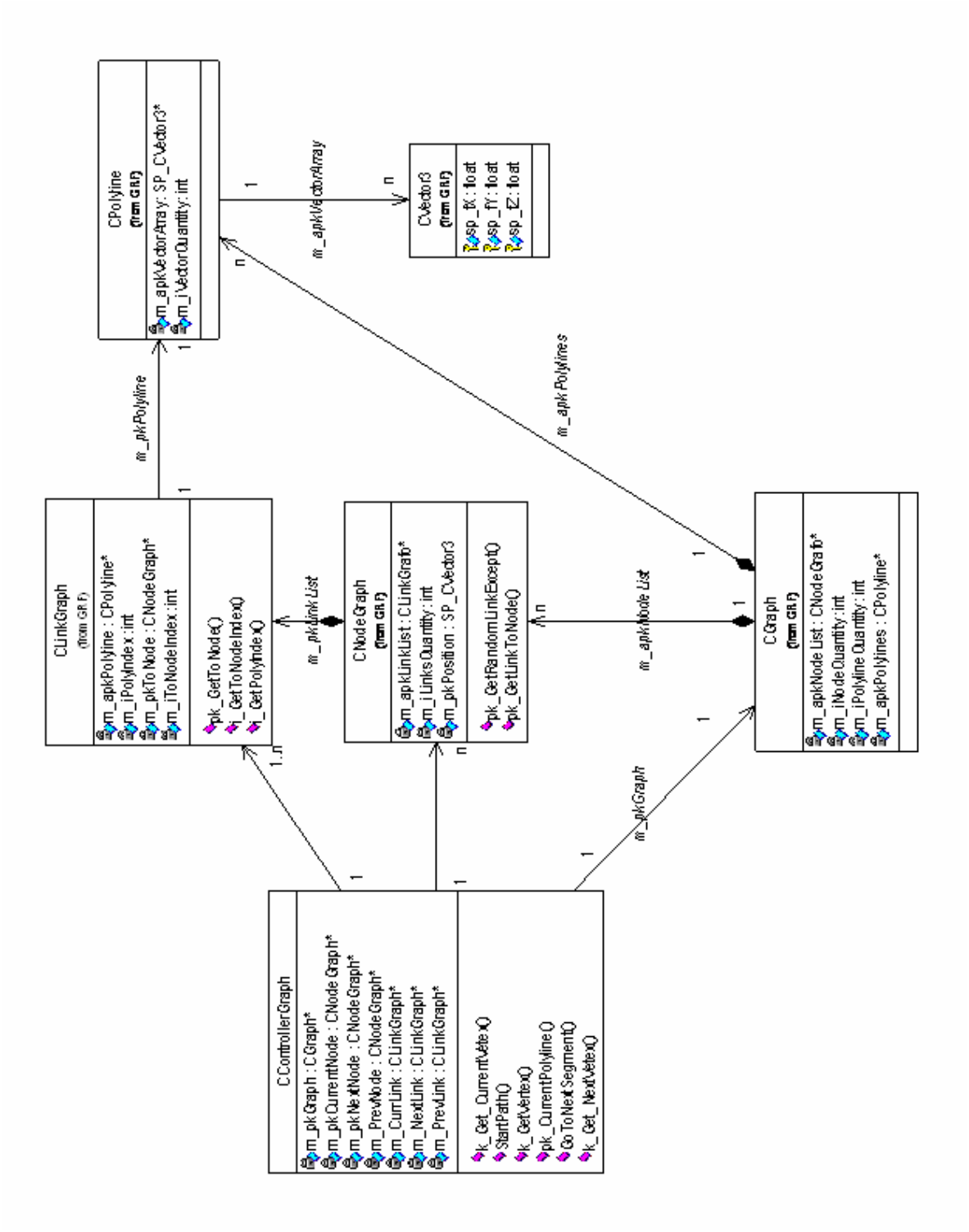


**Figura 25 Diagrama de Clase Diseños “Control de Traslación”.**

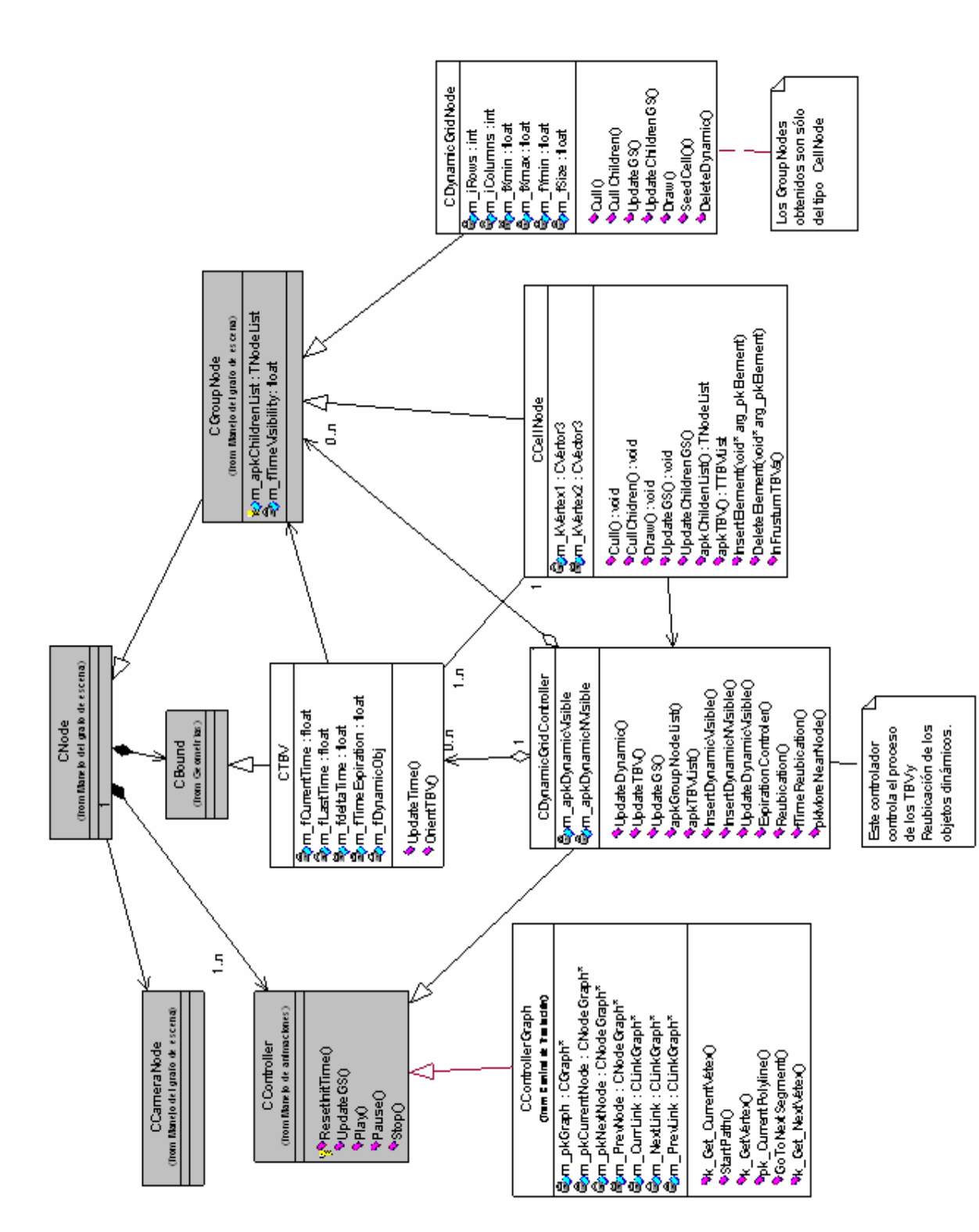


**Figura 26 Diagrama de Clase Diseños “Manipulación de Visibilidad”.Las clases color gris son las pertenecientes a la herramienta.**

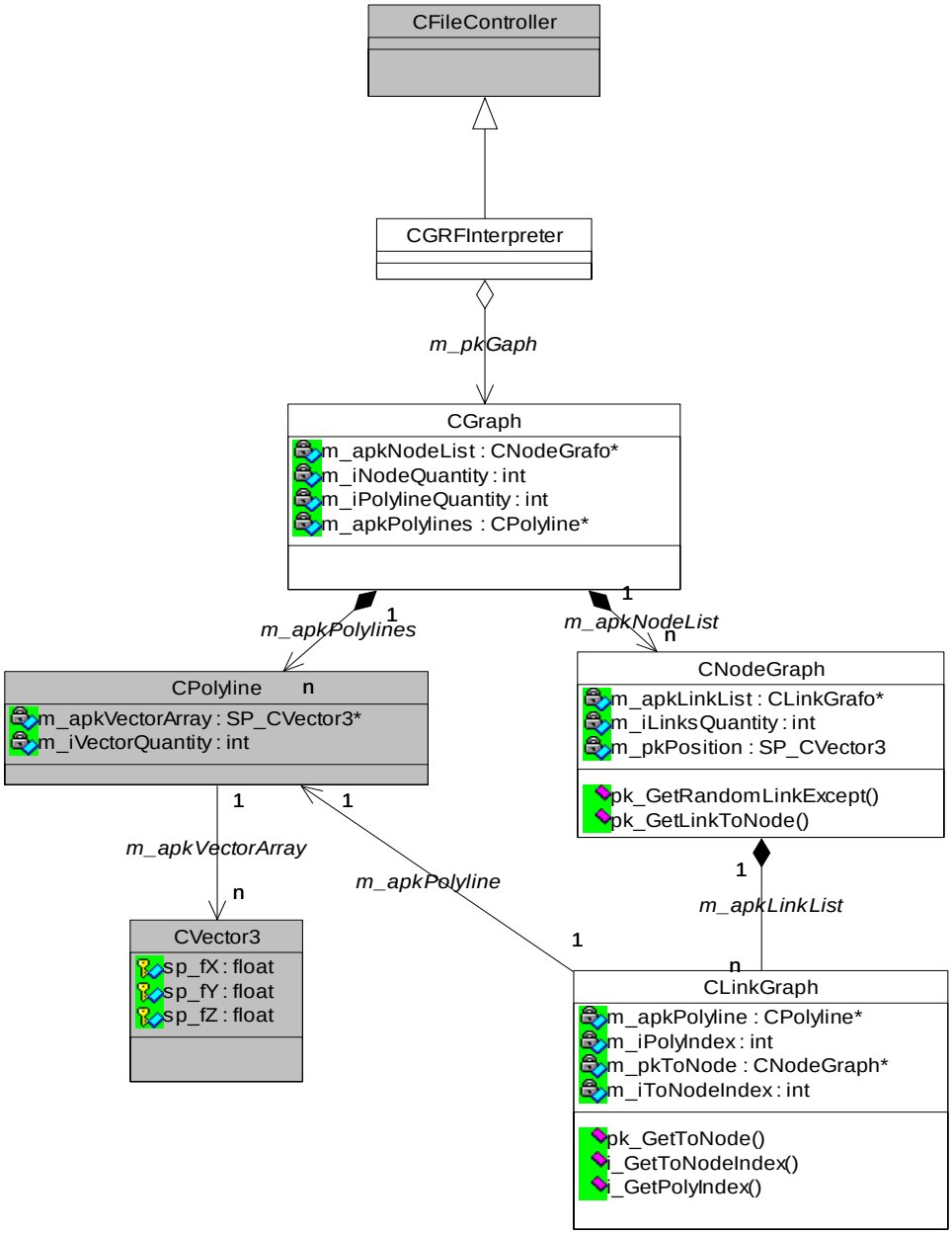


**Figura 27 Diagrama de Clase Diseños “GRF” las clases. Las clases de color gris son las pertenecientes a la herramienta.**

### 4.7.3 Diagramas de Secuencias

A continuación se presentarán los diagramas de secuencia agrupado por casos de uso.

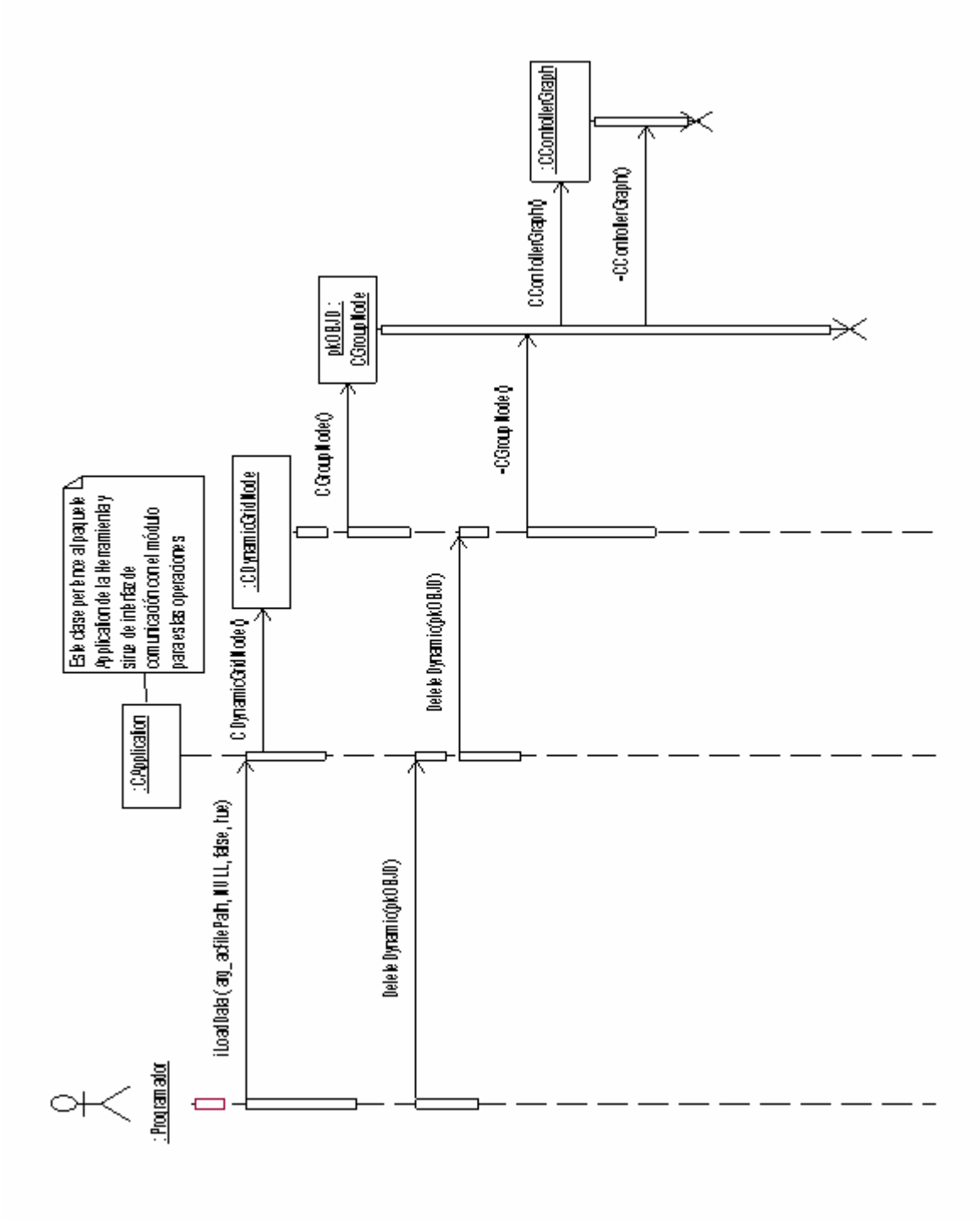


**Figura 28 Diagrama de Secuencia del Caso de Uso “Gestionar Objeto Dinámico”.**

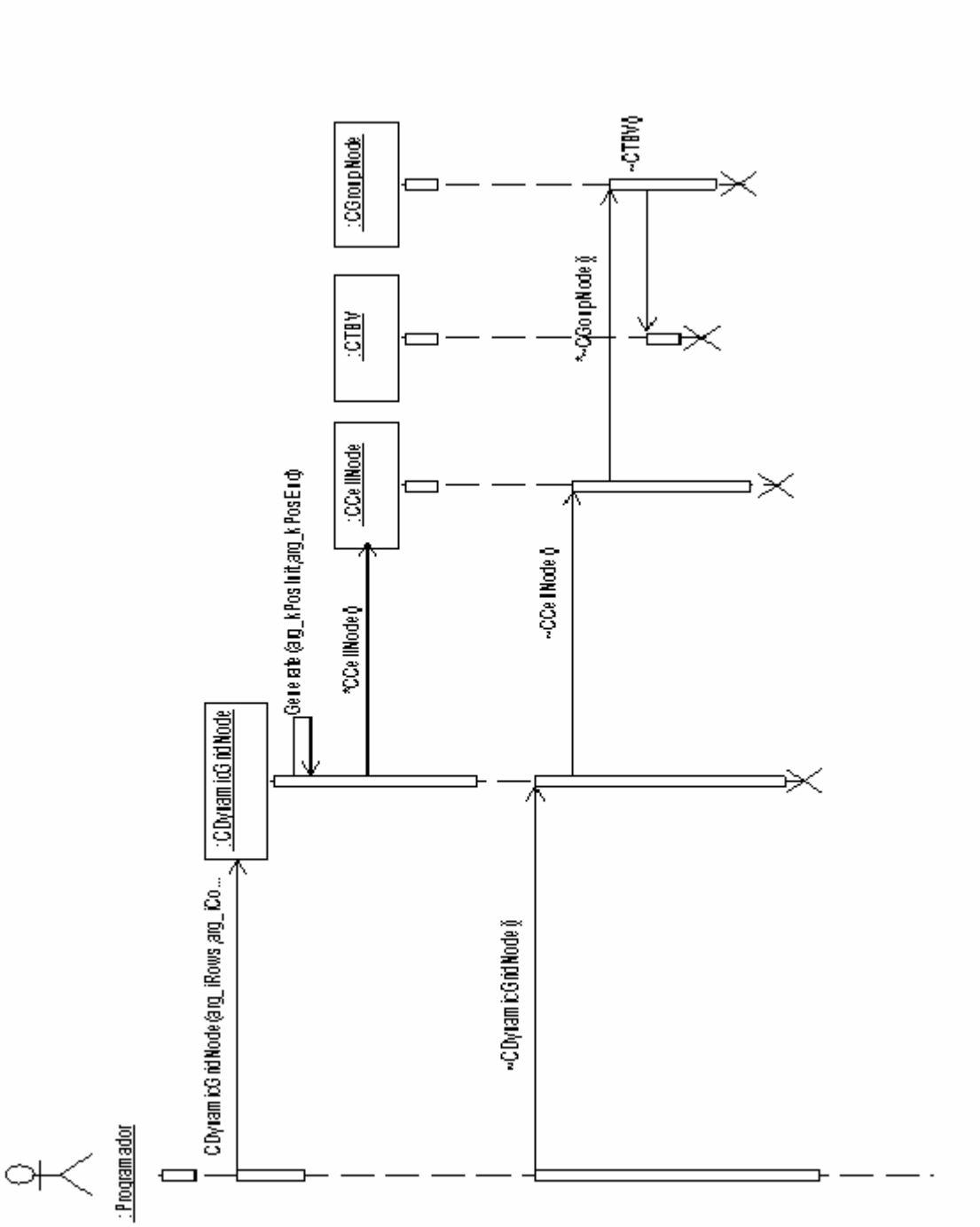


**Figura 29 Diagrama de Secuencia del Caso de Uso “Gestionar Rejilla Regular”.**

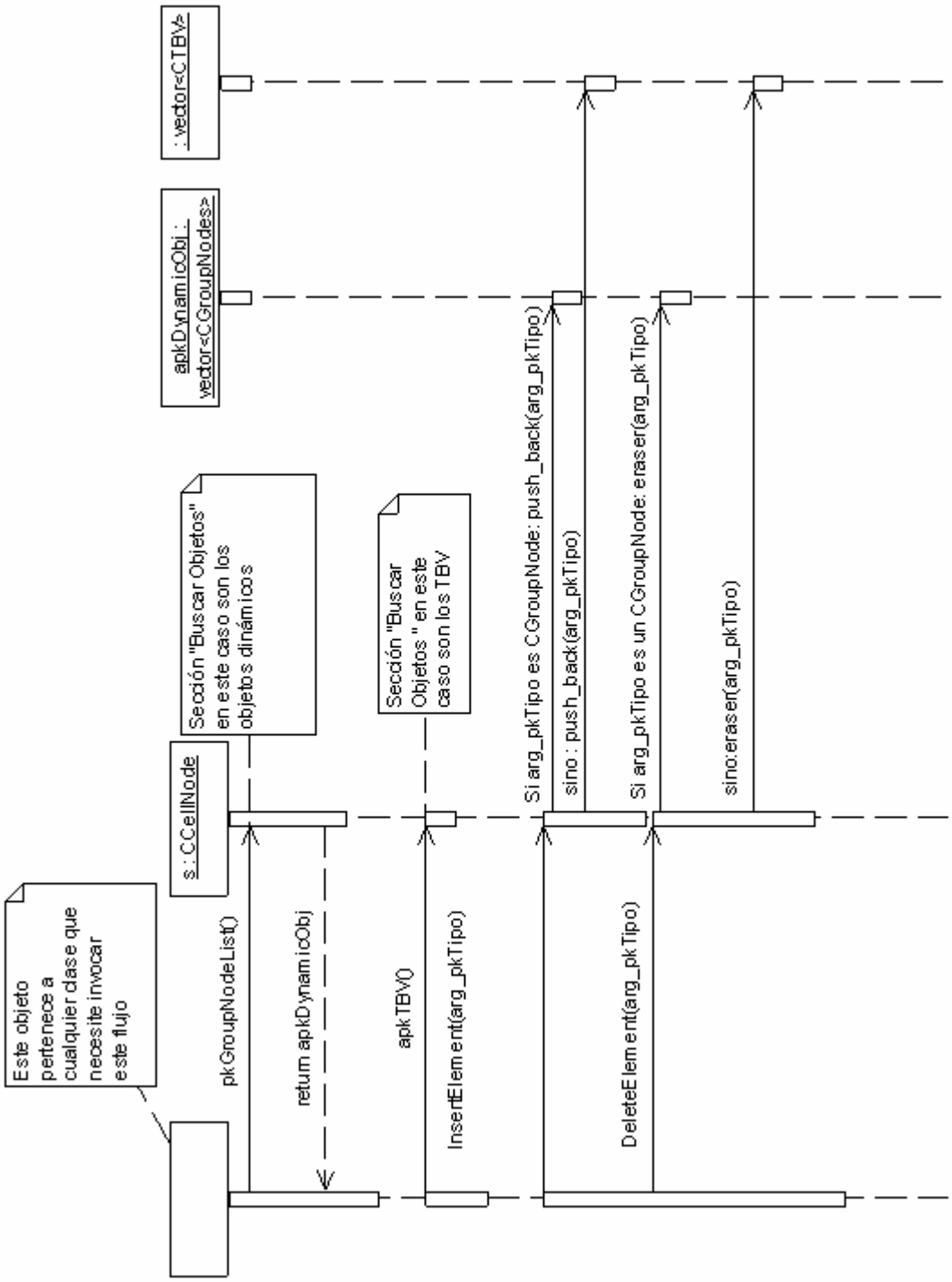


**Figura 30 Diagrama de Secuencia del Caso de Uso “Gestionar Celda”.**

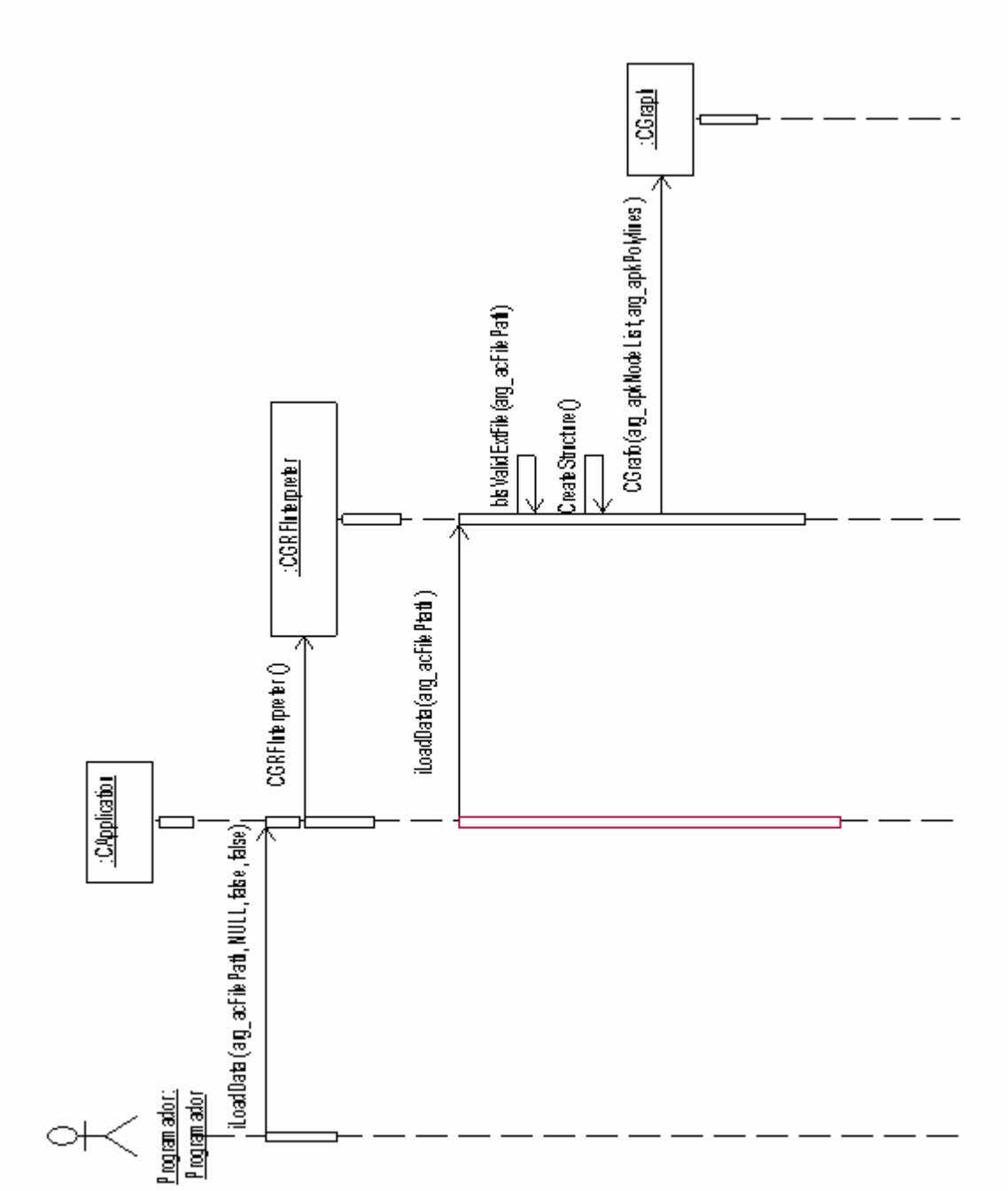


**Figura 31 Diagrama de Secuencia del Caso de Uso “Cargar Grafo del Entorno”.**

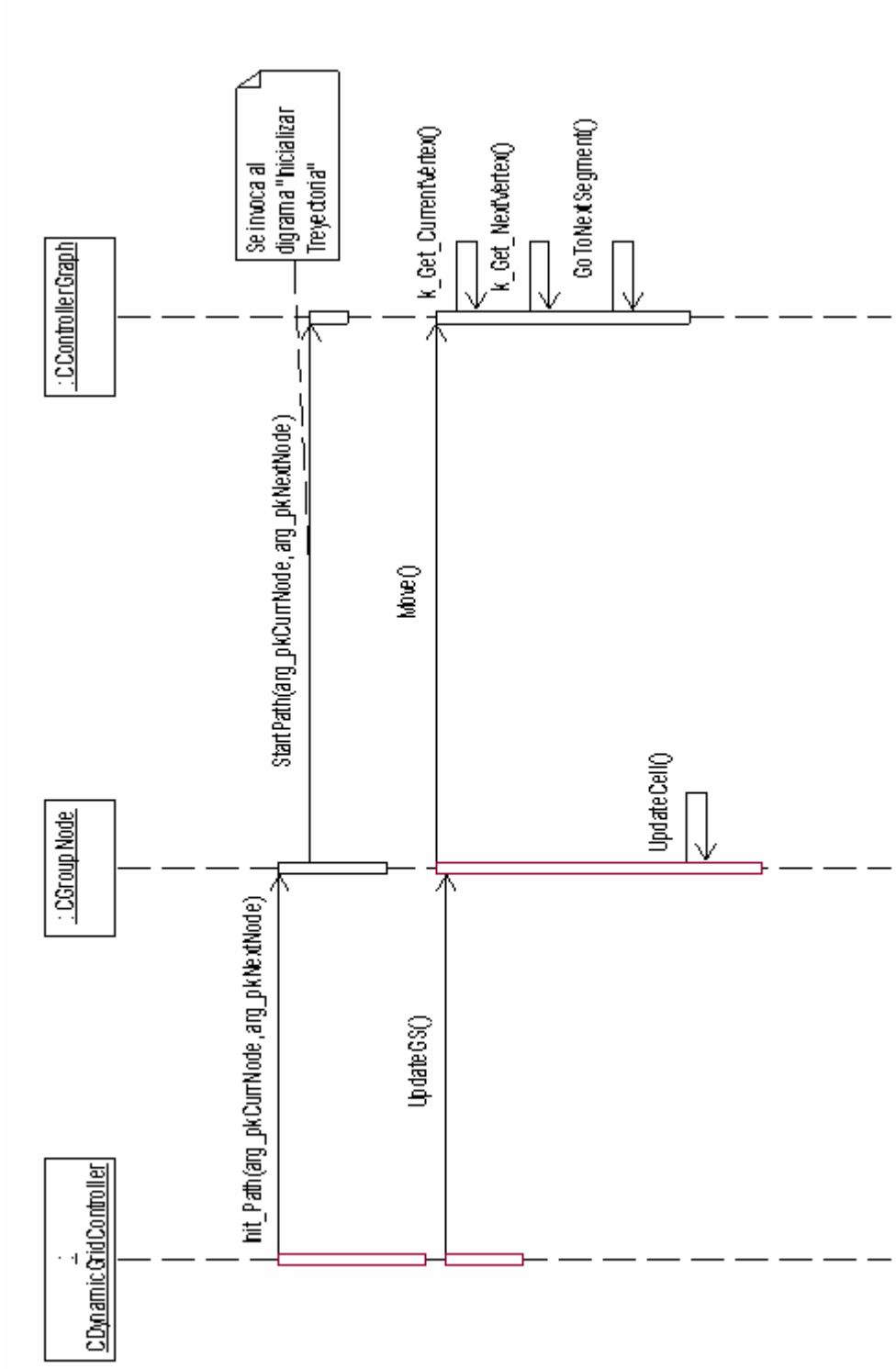


**Figura 32 Diagrama de Secuencia Principal del Caso de Uso “Trasladar Objeto Dinámico”**

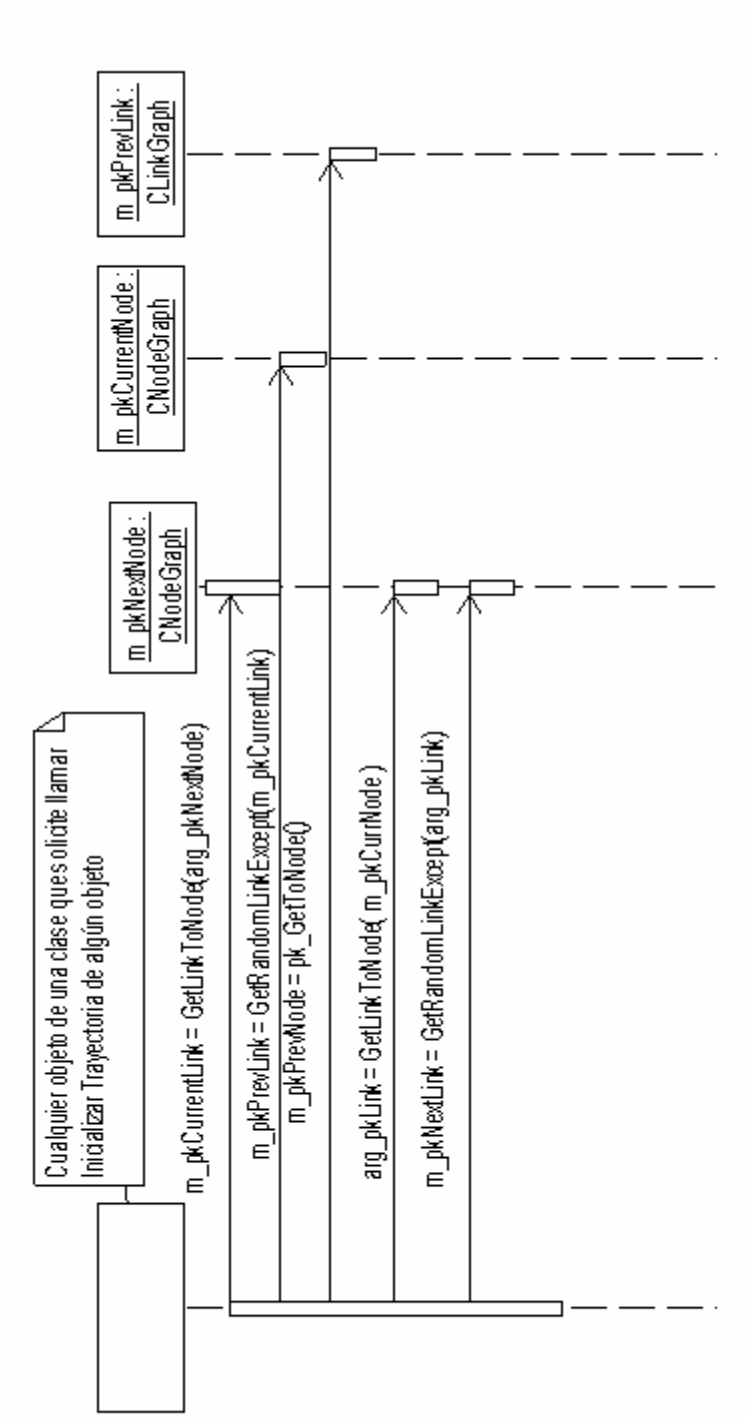


**Figura 33 Diagrama de Secuencia “Inicializar Trayectoria” del Caso de Uso Trasladar Objeto Dinámico”.**

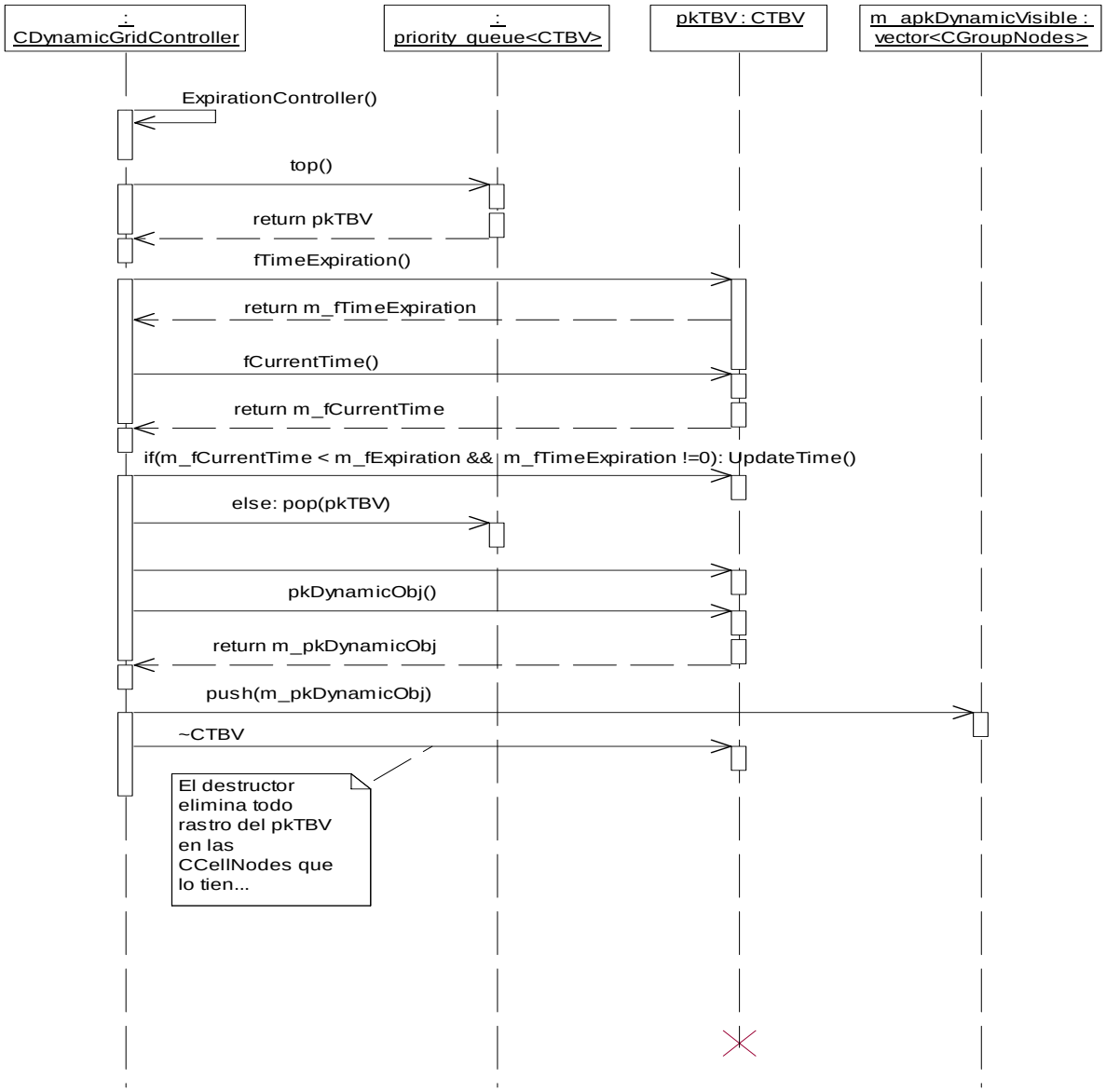


**Figura 34 Diagrama de Secuencia del Caso de Uso “Controlar Expiración TBV”.**

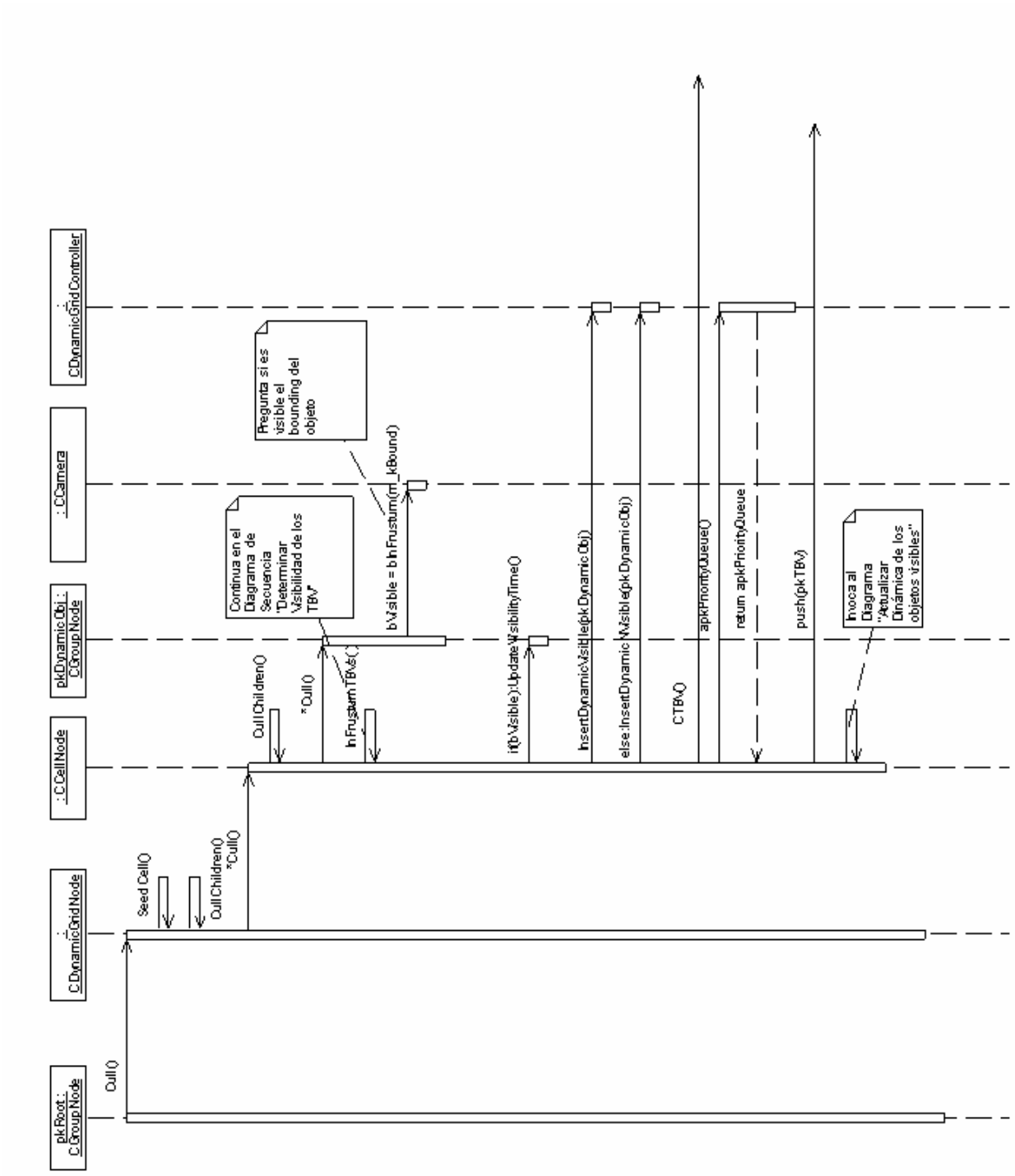


**Figura 35 Diagrama de Secuencia Principal del Caso de Uso “Determinar Visibilidad”.**

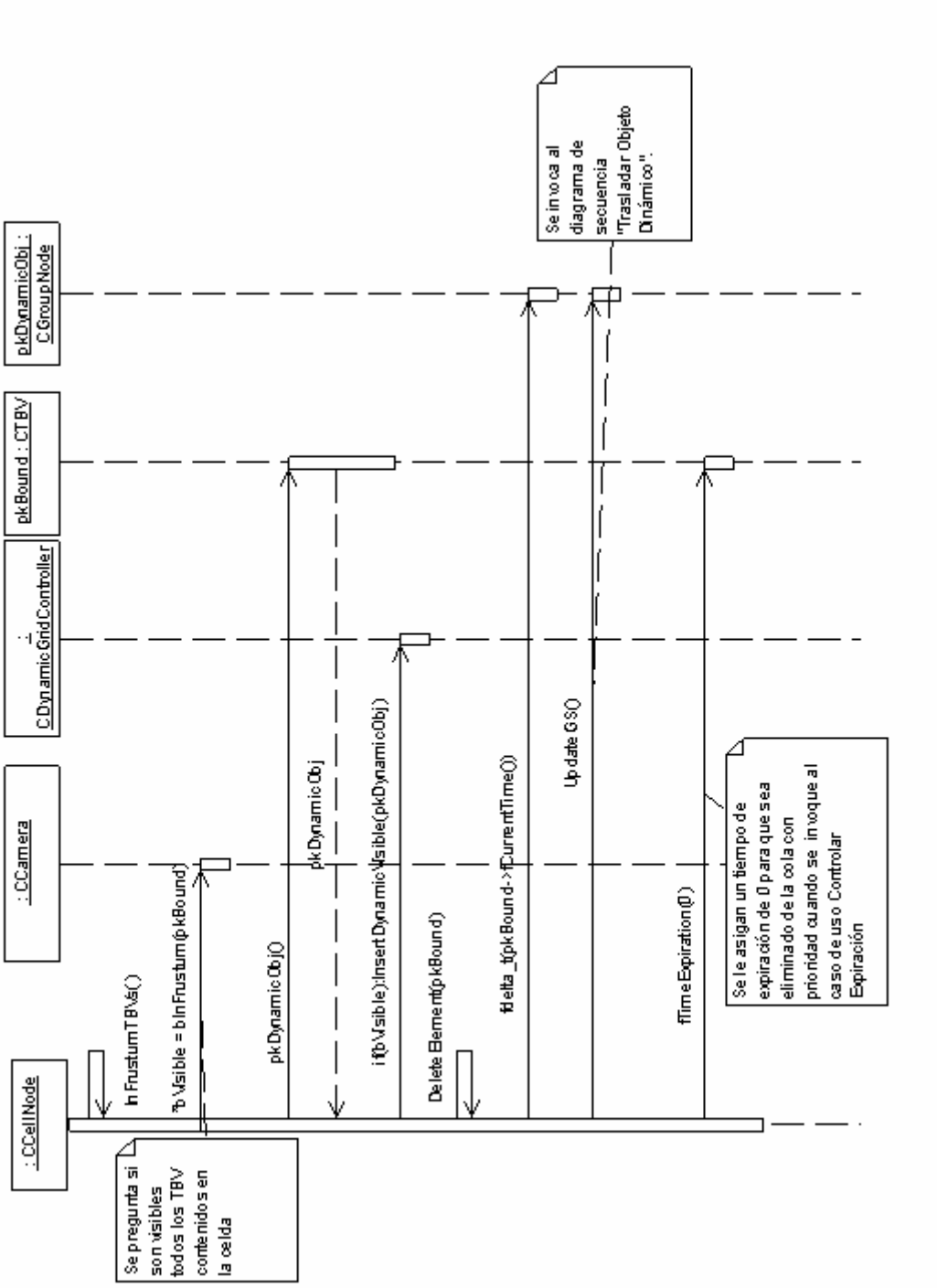


**Figura 36 Diagrama de Secuencia “Determinar Visibilidad de los TBV” del Caso de Uso Determinar Visibilidad**

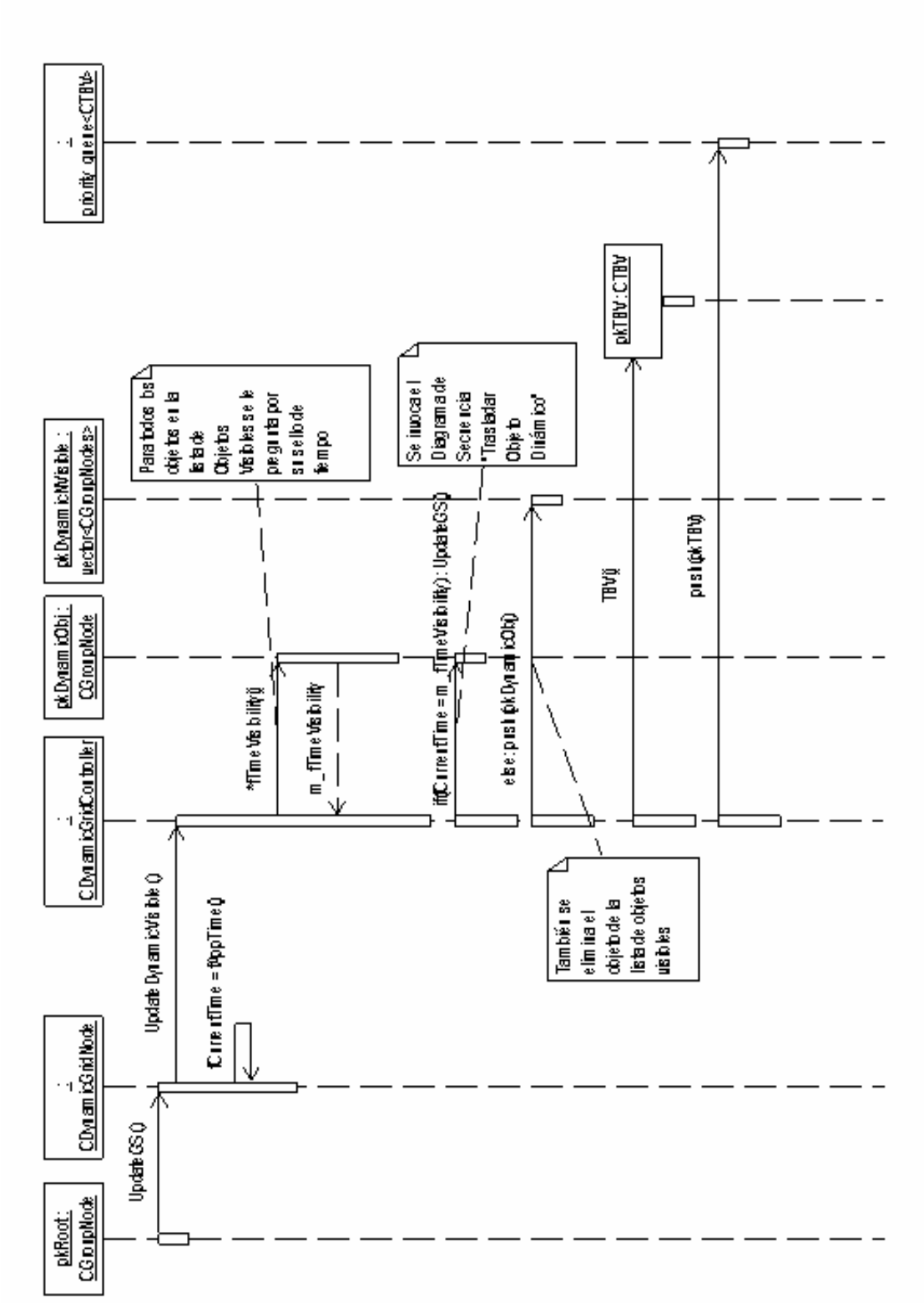


**Figura 37 Diagrama de Secuencia “Actualizar dinámica de los objetos visibles” del Caso de Uso Determinar Visibilidad.**

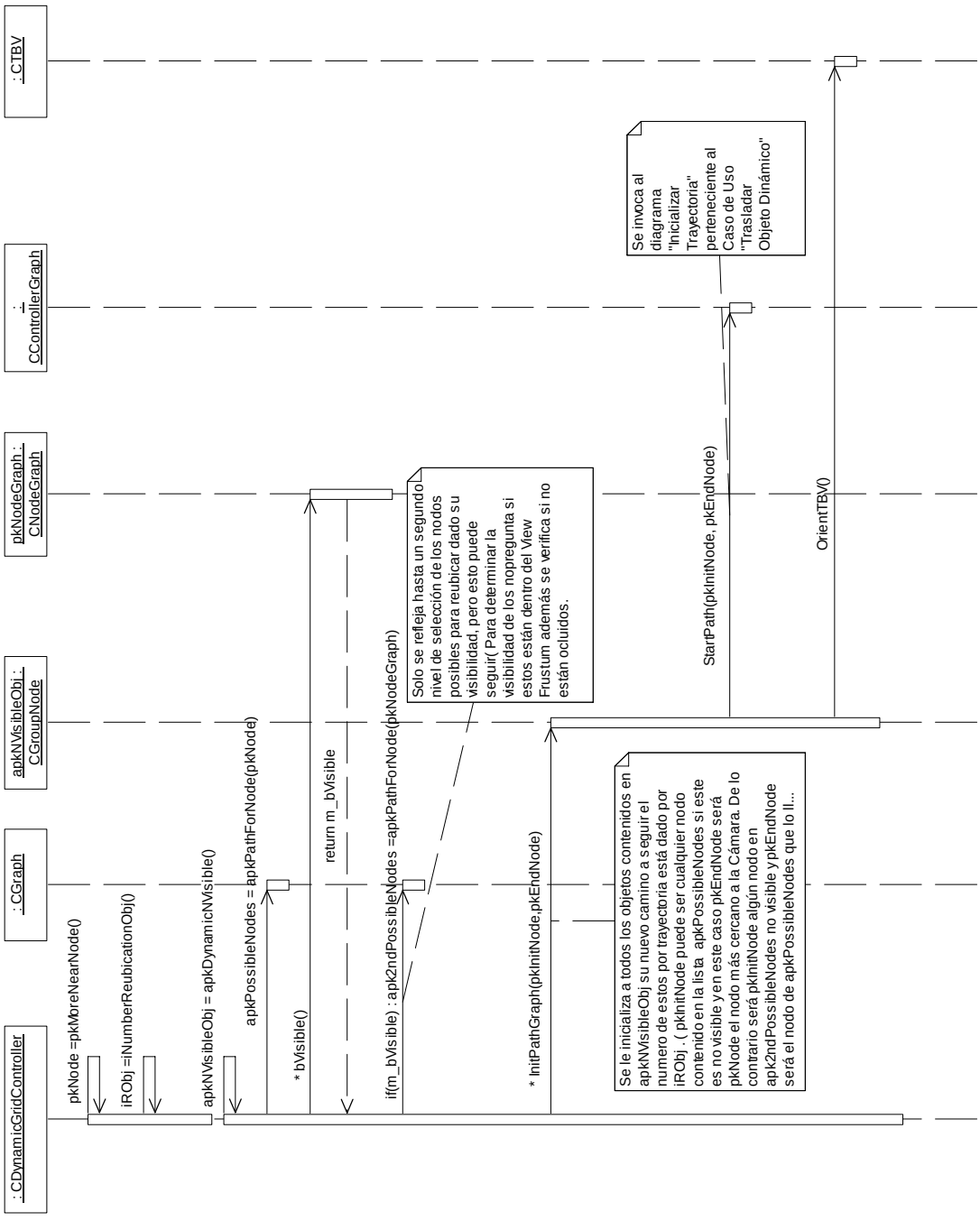


**Figura 38 Diagrama de Secuencia del Caso de Uso “Reubicar Objetos Dinámicos”.**

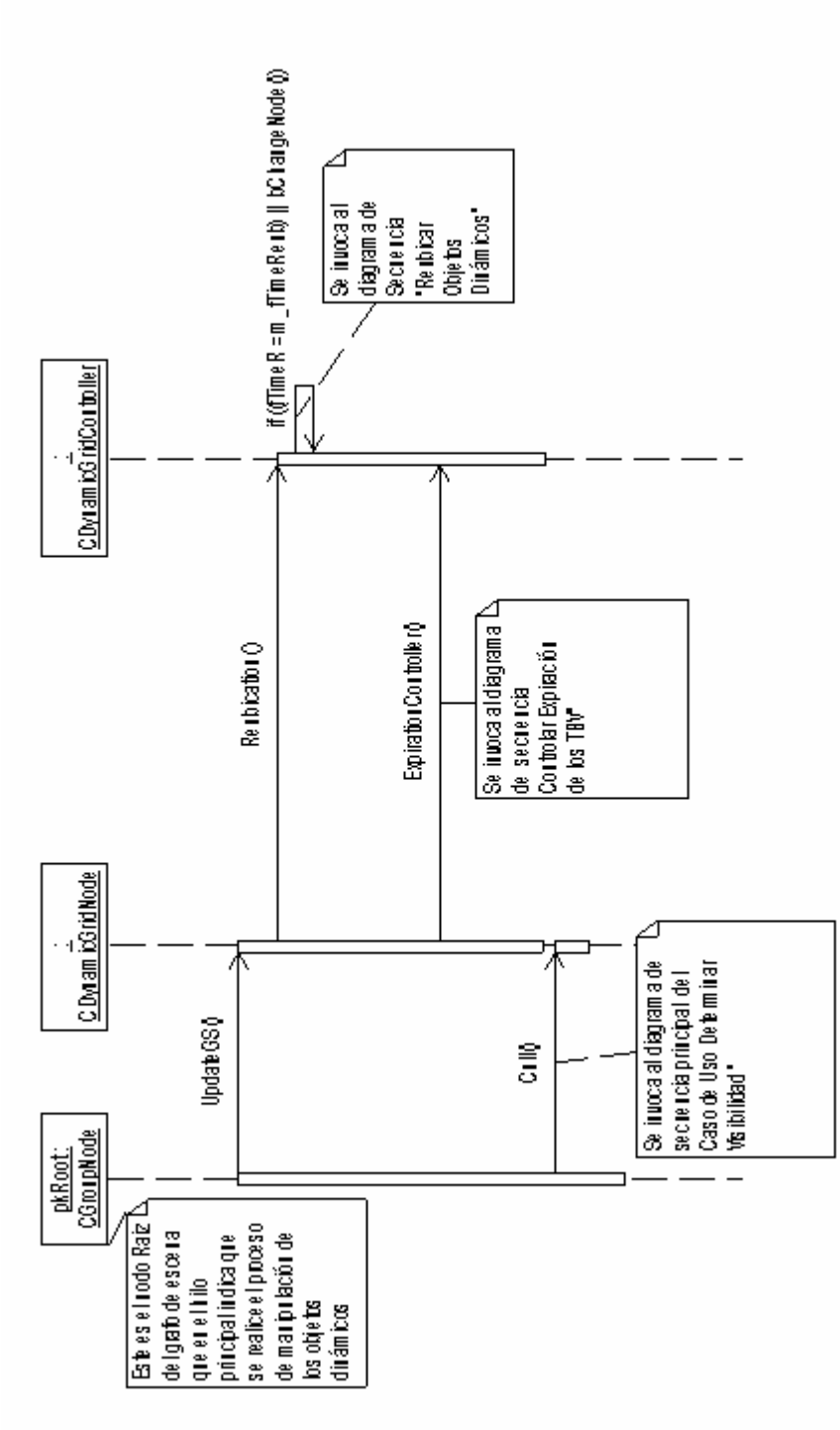


**Figura 39 Diagrama de Secuencia del Caso de Uso “Procesar Objetos Dinámicos”.**

## 4.8 Implementación del Sistema

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

### 4.8.1 Estándares de Codificación.

El código seguirá el mismo estándar de la Herramienta. Este sigue algunos estándares propuestos por el grupo de desarrollo (respetando los estándares de codificación para C++ .Está programarado en inglés, debido a que las palabras son simples, no se acentúan y es un idioma muy difundido en el mundo informático. El conocimiento de los estándares seguidos para el desarrollo de la misma permitirá un mayor entendimiento del código, y es una exigencia de los autores 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

**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 **E**Node**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 iteradotes STD:***
vector<> **T**Name**List**;
TNameList::iterator **T**Name**ListIter**;

map<> **T**Name**Map**;
TNameMap::iterator **T**Name**MapIter**;

multimap<> TNameMultiMap;
TNameMultiMap::iterator TNameMultiMapIter;

**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 “m_” (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 bVarName;
int iName;
unsigned int uiName;
float fName;
char cName;
char* acName; // arreglo de caracteres
char* pcName; // puntero a un char
char** aacName; // bidimensional
char** apcName; // arreglo de punteros
bool m_bMemberVarName; //variable miembro
char gcGlobalVarName; //variable global, no se le antepone “”
short sName;
Instancias de tipos creados:
EMyEnumerated eName;
SMyStructure kName;
CClassName kObjectName;
CClassName* pkName; //puntero a objeto
CClassName* akName; //arreglo de objetos

CClassName* akName; // variable miembro de clase
IMyInterface* pIName; //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);
~CClassName ();

Funciones:

```
bool bFunction1 (…);
int* piFunction2 (…);
CClassName* pkFunction3 (…);
```

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;
}
```

### 4.8.2 Diagrama de componentes

Los componentes del módulo forman parte del paquete de implementación Engine y están contenidos en el interior de los subpaquetes del mismo. A continuación se mostrará las dependencias de estos subpaquetes de acuerdo a los componentes del módulo y seguidamente se mostrará los componentes del módulo a implementar con sus respectivas dependencias. Los paquetes de color gris representan los paquetes de implementación que contienen a los componentes a implementar, que son igualmente grises.

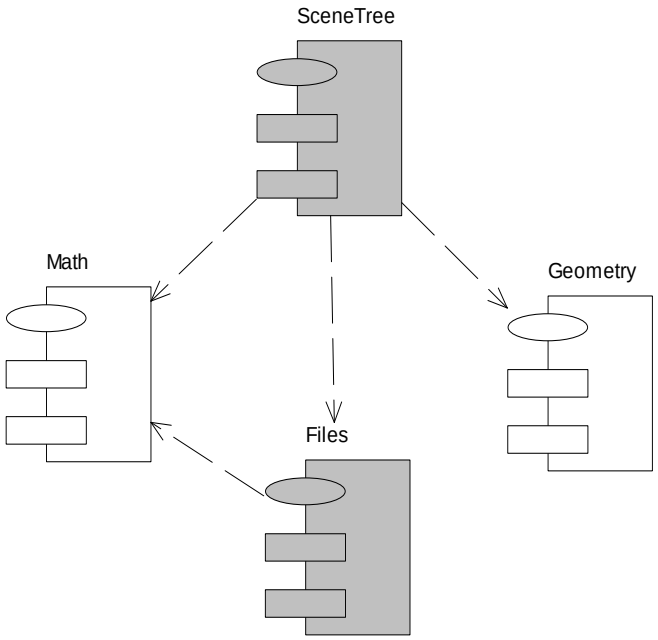


**Figura 40 Diagrama de Subpaquetes .Los Subpaquetes de color gris contienen los componentes a implementar del módulo.**

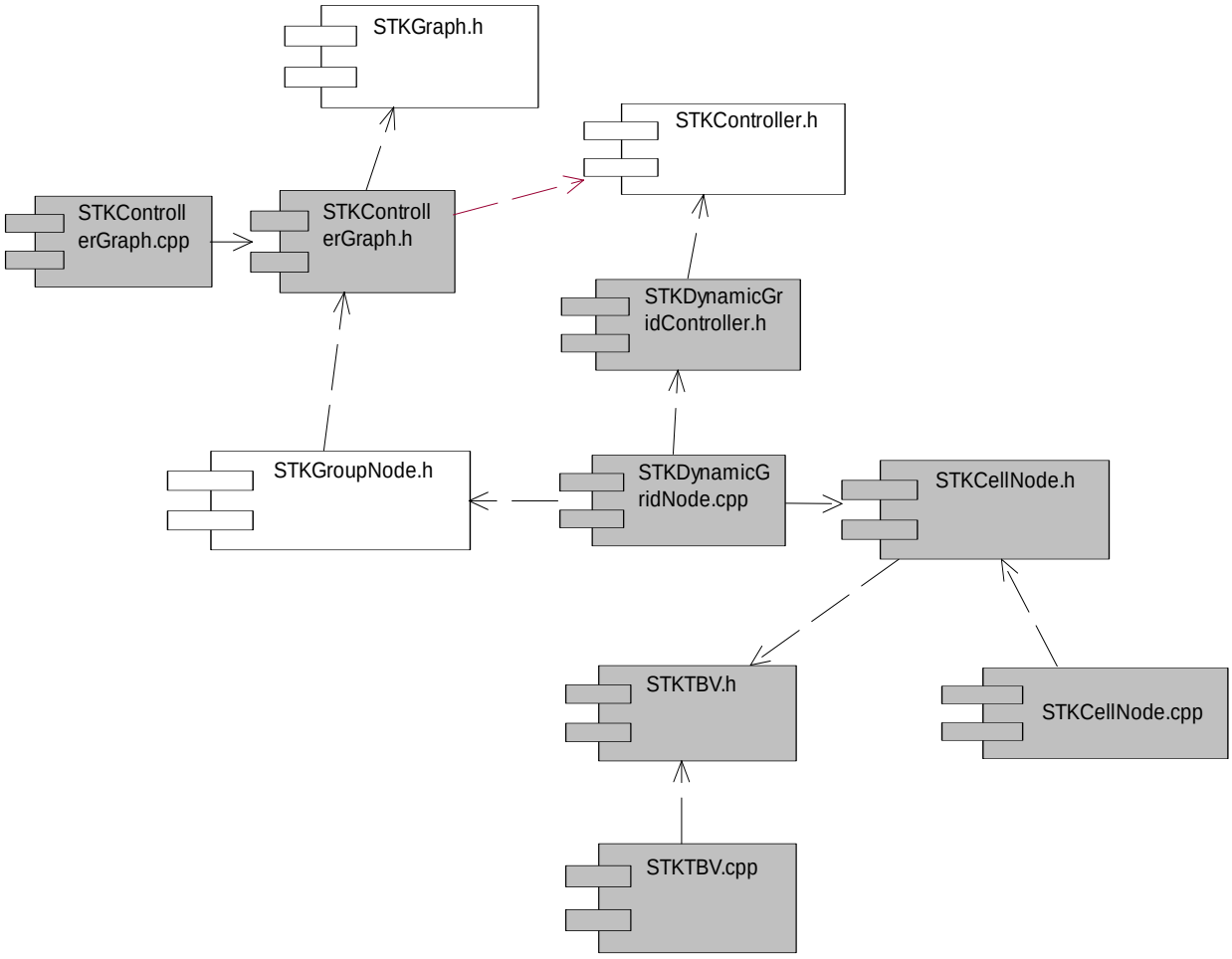


**Figura 41 Diagrama de componentes # 1 "SceneTree". Estos son los componentes que implementará el paquete de diseño Manipulación de Visibilidad y Control de Traslación.**

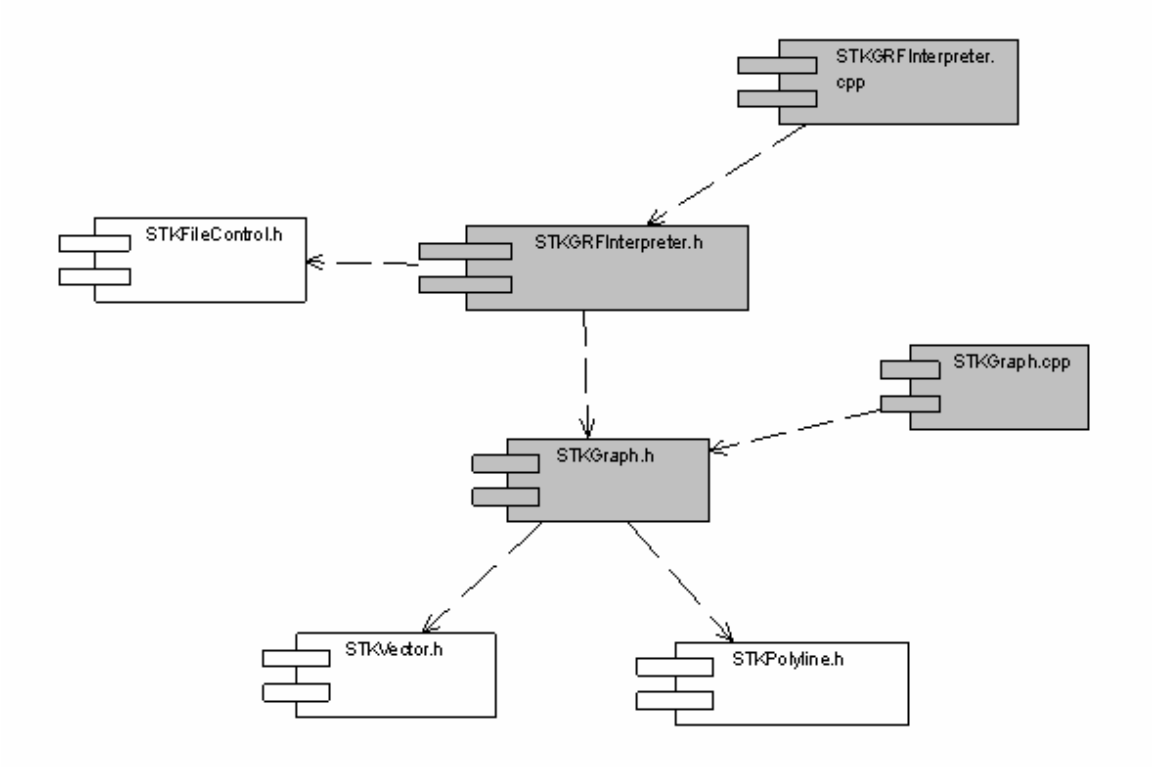


**Figura 42 Diagrama de componentes "Files". Estos son los componentes que implementará el paquete de diseño GRF**

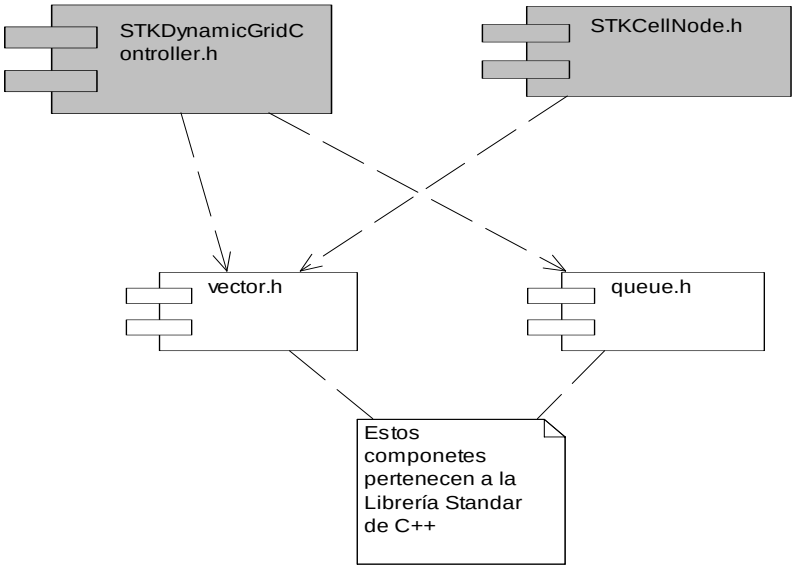


**Figura 43 Diagrama de componentes #2 "SceneTree" donde se deja ver algunas dependencias con otros componentes importantes.**

## Conclusiones

En el presente capítulo se hace una propuesta de diseño del módulo que manipulará a los objetos dinámicos de acuerdo a las especificidades de la herramienta. Para ello quedaron establecidos los requisitos funcionales de éste, y descritos los casos de uso que le permitirá a su futuro usuario obtener los resultados esperados a la hora de manipular los objetos dinámicos.

# Conclusiones

Dando cumplimiento a los objetivos de este trabajo se realizó un estudio de las técnicas y algoritmos para el manejo de objetos dinámicos. En el estudio se analizaron las características, ventajas y desventajas de éstos. Además, a partir de esta investigación se definió el Algoritmos de los TBV para dar solución al problema de visibilidad de una forma eficiente y el Algoritmo de Reubicación para el ahorro de recursos de memoria.

Posteriormente se pasó a la implementación de los mismos por separado demostrando su eficiencia en el ahorro de recursos, y luego se unieron en el Algoritmo TBV+Reubicación para demostrar su compatibilidad y lograr una solución eficiente para el manejo de los objetos dinámicos en escenas urbanas.

Finalmente se estudió la arquitectura de la herramienta “SceneTollkit” y se propuso un diseño del Módulo para el Manejo de Objetos Dinámicos en el cual el Algoritmo TBV+Reubicación fue tomado como solución propuesta a implementar en el módulo.

# Recomendaciones

A este trabajo se le recomiendan los siguientes aspectos:

- Continuar el estudio de las técnicas y algoritmos que van surgiendo para manipular eficientemente los objetos dinámicos de los entornos virtuales urbanos y proponer mejoras para los Algoritmos definidos en este trabajo.
- Agregar la oclusión en el cálculo de visibilidad de los objetos.

- Al fichero .grf, agregarle información sobre la diferenciación de las trayectorias, dado que no debe ser la misma cantidad de objetos a reubicar por nodos, esto debe estar en consecuencia con la densidad necesaria de objetos por zona. Para ello se debe realizar esta definición en la etapa de diseño del entorno virtual.

- Implementar el módulo de clases diseñado en este trabajo.

# Referencias bibliográficas

**[1] Borja Fdez Gauna,Disponible en :**
**http://www.gamedev.net/reference/programming/features/superfrustum**

**[2] ASTLE, D. and K. HAWKING. *OpenGL Game Programming*. USA, Prima Tech Publishing, 2001. p.**

**[3] B.F, N. *Partitioning tree representation and generation form 3D Geometric Models*, 1992.**

**[4] BATAGELO, H. C. and S.-T. WU. *Dynamic scene occlusion culling using a regular grid*, XV Brazilian Symposium on Computer Graphics and Image Processing, 2002. 43-50 p.**

**[5] *Frustum Culling*. Disponible en: http://www.emeyex.com/site/tuts/FrustumCulling.**

**[6] SUDARSKY, O. and C. GOTSMAN. *Dynamic Scene Occlusion Culling*, 1999. 5: 217-223.**

**[7] T, A. and M. V. *dPVS Reference Manual Version 6.10*, 2005. [Disponible en: http://www.hybrid.fi/dpvs**

**[8] VARVANA, M. *Dynamic Scene Occlusion Culling*, 2003. [Disponible en: http://www.tml.tkk.fi/Opinnot/Tik-111.500/2003/paperit/VarvanaMyllarniemi.pdf**

**[9] N.Greene,M.Kass and G.Miller,*Hierarchical Z-Buffer Visibility*, 2001. [Disponible en: http://www.cs.princeton.edu/courses/archive/spr01/cs598b/papers/greene93.pdf**

**[10] Disponible en :** http://www.renderware.com/dpvs.asp.

**[11] Disponible en** :http://www.hybrid.fi/

**[12] *Bresenham's line algorithm*, Disponible en :**
**http://en.wikipedia.org/wiki/Bresenham's_line_algorithm**

**[13] *VALENCIA, U. D. Sistemas de Visualización en Tiempo Real. Visualización por Hardware, 2004. p.***

**[14] CHOVER, M. *Informática Gráfica*. 2004. p.**

**[15]** **AIRLEY, J.,** ***Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations,*** **1991.**

**[16]** **MONTOTO, J. I. G.; D. A. RODRÍGUEZ,** ***et al.*** **VISIBILIDAD POR REGIONES EN ZONAS URBANAS PARA SIMULADORES DE CONDUCCION, 2004.**

# Apéndices

## Glosario de abreviaturas

**BSP-Tree :** Árbol de Partición Binaria del Espacio.

**TBV** : Temporary Bounding Volumen (Volumen Frontera Temporal)

**LCA** : *Least Common Ancestro* (Mínimo Común Antecesor)

**PVS** : Potentially Visible Set( Colección Potencialmente Visible) se refiere al conjunto de objetos que son candidatos a ser visibles en el actual *frame*.

## Glosario de Términos

**B:**

*bounding volume* : volumen frontera

*Bounding Spheres* : esferas Fronteras

*Bounding Boxes* : cajas fronteras.

***C:***

*celda***:** cuadrícula que forma parte de una Rejilla Regular.

**F**:

*frame* : cada una de las imágenes que componen una animación.

***H:***

*hierarchical view frustum culling* : selección de visibilidad utilizando estructuras jerárquicas por ejemplo: usando el Octree.

**O:**

*occlusion culling :* selección por oclusión

**R:**

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

**S:**

*sensible a la salida**:*** es una característica de los algoritmos que demuestra como un algoritmo independientemente del tamaño de su entrada sus resultados expresado en los frame por segundo será aproximadamente el mismo.

**V:**

*voxel : volumen de encierro en forma cúbica en el caso del Octree y en la Rejilla Regular es el equivalente a una celda.*

view-frustum **:** Sección del espacio 3D o 2D visible por la cámara(Ver epígrafe 1.3)

**Z:**

*Z-buffer* **:** buffer de profundidad.

# Índice de figuras y tablas

## Índice de figuras

## Índice de Tablas