**UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS**

**FACULTAD 5**

# Visualización de Sistemas de Realidad Virtual
## Optimización por Niveles de Detalles

**Trabajo para optar por el Título Ingeniero en Ciencias Informática**

**Autores:** Katherine Gómez Cuello

Ariangna Garcés Gilart

**Tutor:** Ing. Yanoski Camacho Román

Ciudad de la Habana
2007

DECLARACIÓN DE AUTORÍA

Declaramos que somos las únicas autoras de este trabajo, y autorizamos al Proyecto Herramientas de Desarrollo para Sistemas de Realidad Virtual de la Facultad 5 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Autoras:

Katherine Gómez Cuello

_____

Ariangna Garcés Gilart

_____

Tutor:

Yanoski Camacho Román

_____

DATOS DE CONTACTO

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

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

# Dedicatoria

*A mis padres por su apoyo en la distancia.*
*A mi hermano por la confianza.*
*A David, por el amor en los días más duros.*

*Katherine.*

*A mi mamá y al Nene por estar siempre.*
*A  Lázaro por ser mi apoyo y a mi familia.*

*Ariangna.*

# Agradecimientos

A  Yanoski, nuestro tutor, por tantas horas dedicadas, sin él no hubiésemos llegado.

A  Fernando por las dudas que aclaró en su poco tiempo libre.

A  Karel por los modelos 3D.

A todos los que estuvieron pendientes de nuestros pasos, cerca o lejos.

Gracias!

# Resumen

Una meta perenne -aunque no la única-, de los gráficos de computadora es producir evocaciones visuales de mundos virtuales que parezcan verdaderas.

Este trabajo aborda el diseño de una solución para optimizar la visualización a través de los niveles de detalle, garantizando que las primitivas que alimentarán los gráficos en la tubería de visualización lleven el mínimo de polígonos necesarios, eliminando aquellos que según su distancia al punto de visión, tengan una proyección en la imagen final substancialmente menor que la resolución de píxeles de la exhibición.

Para alcanzar esta meta se trabaja en la investigación y desarrollo de algoritmos eficientes para aplicar niveles de detalles en los entornos virtuales, y se realiza una exposición de los conceptos, algoritmos, y estructura de datos para el trabajo con dichos niveles. El módulo resultante de esta investigación, permitirá mayor calidad en la visualización de las aplicaciones finales de los proyectos de realidad virtual que hagan uso de este.

# Índice de contenido

# Índice de tablas



# Índice de figuras

# Introducción

Con el surgimiento y desarrollo de las técnicas de Realidad Virtual (RV), la Informática Gráfica ha dado un salto en la calidad de sus aplicaciones en cuanto a interactividad y realismo, dando paso al surgimiento de los Sistemas de Realidad Virtual (SRV), sistemas orientados fundamentalmente al desarrollo de habilidades en la manipulación de equipos mediante una ambientación digital.

Con un gran acercamiento a la realidad y un considerable ahorro de recursos de estudio y entrenamiento, los SRV dejan ver sus beneficios en la industria del entretenimiento (vídeo-juegos); en la visualización científica de datos y de fenómenos no perceptibles por el ojo humano; en la medicina (realización de prótesis, intervención médica a niveles celulares, ingeniería genética, teleoperaciones, virtuoterapia,…); en la meteorología (simulaciones de tormentas eléctricas, impactos geológicos de volcanes en erupción…); en la defensa (entrenamiento de soldados en el desarrollo de habilidades en la manipulación de los medios de combate, y en la toma de decisiones adecuadas en situaciones críticas); etc. [4].

En Cuba, una de las empresas que ha impulsado el desarrollo de los SRV es el “Centro de Investigación y Desarrollo #2” (CID2), conocida en el mercado como SIMPRO (Simuladores Profesionales), la cual cuenta con numerosos premios a las investigaciones y calidad de sus productos desde su fundación.

La Universidad de las Ciencias Informáticas (UCI), proyecto que desde sus inicios ha estado dirigido a la formación de profesionales y a la producción de software, tiene entre sus líneas de investigación-producción el tema de la Realidad Virtual, por lo que en asociación con SIMPRO se han desarrollado varios proyectos con el fin de desarrollar SRV en la Universidad y llegar a establecer un Centro de Realidad Virtual de referencia nacional.

El siguiente organigrama (figura 1) muestra la organización básica de los Proyectos de Realidad Virtual, los cuales se encuentran en cooperación bajo un macro-proyecto único:

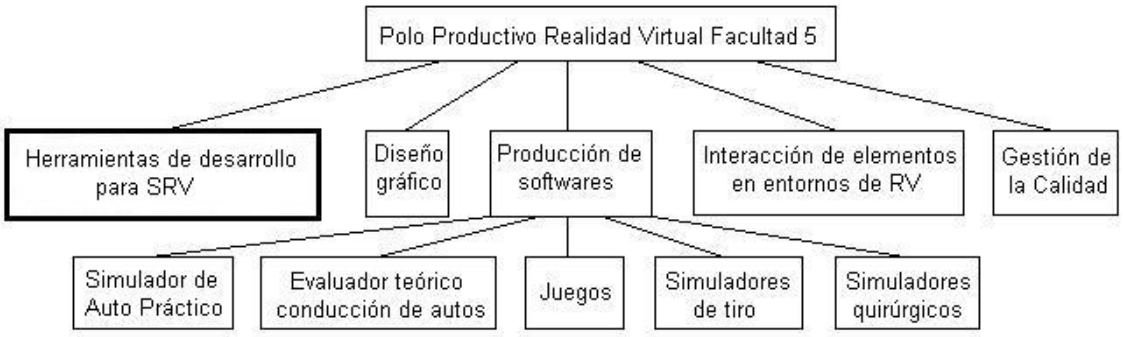



**Fig. 1 Organigrama de los Proyectos de Realidad Virtual**

Entre estos proyectos se encuentra el de Herramientas de desarrollo para Sistemas de Realidad Virtual (cuadro destacado en la figura 1), cuya misión es producir herramientas con las funcionalidades comunes a los demás proyectos, de manera que se organice el trabajo y se les brinden herramientas a los programadores de las aplicaciones finales, haciendo así más ágil su trabajo. Herramientas de este tipo hay muchas y muy eficientes en Internet, pero no siempre cumplen con los requerimientos de los proyectos, o son demasiado caras por lo que provocarían una dependencia monetaria y necesidad de actualizaciones de los proveedores. Resulta entonces conveniente tener herramientas de trabajo propias que respondan a los intereses de los proyectos de la facultad.

Este proyecto ya tiene una versión inicial de una herramienta básica, que ha respondido a los requerimientos de los simuladores producidos hasta el momento, pero aún le faltan algunos módulos importantes para llegar a ser una herramienta profesional.

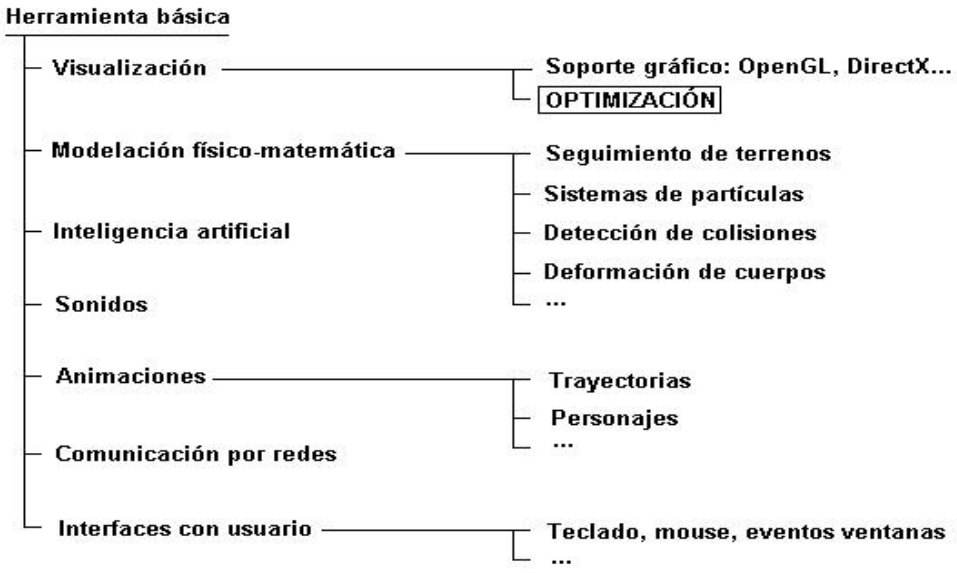



**Fig. 2 Módulos de la Herramienta Básica**

Como se puede observar en la figura 2, una de las responsabilidades de la biblioteca básica, y de la cual depende en gran medida el realismo en cuanto a la calidad del entorno que se vaya a visualizar, es la **optimización de la visualización**, que trata de lograr, a través de estructuras de datos espaciales y técnicas y algoritmos para la visualización eficiente, que se muestre una cantidad adecuada de fotogramas por segundo (FPS) de manera que la visualización le resulte fluida a los usuarios finales, independientemente de si la carga de objetos en la escena es muy grande, o de la cantidad de procesos o cálculos paralelos que se necesiten hacer durante la visualización.

Para lograr una visualización eficiente, típicamente se siguen tres pasos que constituyen el *visibility pipeline* (tubería de visualización), estos son: 1) selección por prisma de visión, 2) selección por oclusión, 3) selección por nivel de detalle.

En la biblioteca básica se tienen acopladas las dos primeras técnicas y la visualización es bastante fluida (se sobrepasa actualmente los 25 FPS mínimos necesarios, incluso los 60 deseados), pero es requerimiento de los proyectos finales que se incluyan determinados efectos visuales de considerable peso (luces, sombras, brillo, reflejos, fuego…), así como cálculos físico-matemáticos que deben disminuir considerablemente la calidad de la visualización, por tanto, paralelo a la implementación de estos nuevos

requerimientos, se necesita incorporar otras técnicas de visualización de manera que la biblioteca esté preparada para soportar los nuevos módulos.

Entonces, ¿cómo obtener a través de un módulo de optimización por niveles de detalles la visualización más eficiente para la herramienta básica del proyecto Herramientas de Desarrollo de Sistemas de Realidad Virtual?, es un **problema a resolver**, y constituye tema de este trabajo de diploma.

La optimización por LOD puede hacerse por software mediante estructuras de datos y algoritmos, o aprovechando determinadas funcionalidades del hardware gráfico de las PC, o ambas. El **objeto de estudio** de esta tesis es la optimización de la visualización para sistemas de realidad virtual, y el **campo de acción**, los algoritmos para la optimización por niveles de detalles.

Se propone como **objetivo general** implementar un módulo funcional para la optimización por niveles de detalles acoplable a la estructura actual de la herramienta básica del proyecto Herramientas de Desarrollo de Sistemas de Realidad Virtual.

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

- Investigación de las principales técnicas de visualización de los motores de Sistemas de Realidad Virtual.
- Investigación de la optimización por niveles de detalles, parámetros de los entornos virtuales que se modifican o se optimizan, algoritmos y tendencias actuales.
- Estudio de la arquitectura y características de la Biblioteca Básica desarrollada por el proyecto Herramientas de Desarrollo de Sistemas de Realidad Virtual.
- Analizar y diseñar una arquitectura de clases que responda a los objetivos planteados.

Como **idea a defender** planteamos que un módulo de optimización por niveles de detalles acoplado a la herramienta mejorará considerablemente la calidad de la visualización de los entornos virtuales creados, independientemente de si la carga de

objetos en la escena es muy grande, o de la cantidad de procesos o cálculos paralelos que se necesiten hacer durante la visualización.

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

En el capítulo inicial, “Fundamentación Teórica”, a través de investigación por búsqueda bibliográfica se exponen las características de los SRV, así como los principales algoritmos existentes en el mundo para la selección de los niveles de detalle y la deformación de mallas.

En el capítulo 2, “Descripción de la solución propuesta”, abarca todo lo relacionado a las características técnicas de la solución planteada, además de las reglas que regirán el negocio. Se realizará la captura de requisitos, y los modelos de dominio y casos de uso del sistema.

Ya en el capítulo 3, “Análisis y diseño del sistema”, se muestra el diseño del sistema propuesto. Se representan las clases del diseño y los diagramas de interacción.

Para el capítulo 4, “Implementación del sistema” se muestra los diagramas de componentes.

En último lugar, aparecen los glosarios de abreviaturas y términos para comprender los vocablos técnicos usados en el desarrollo de la investigación.

# 1

# Capítulo 1 Fundamentación Teórica

## Introducción

Los requerimientos computacionales y de almacenamiento requeridos para las escenas usadas en aplicaciones de realidad virtual exceden por mucho la capacidad del hardware gráfico de visualización. Estas escenas tienen una estructura compleja y su despliegue requiere un gran número de polígonos, aún así cuando se considera solo la porción de la escena que es visible para un marco dado. Tan temprano como en 1776, Clark sugirió el uso de versiones más simples de la geometría para objetos que tuvieran menos importancia visual, tales como aquellos que se encontraran lejos del observador. [1]. Estas simplificaciones son llamadas Niveles de Detalle (LODs).

La meta de la simplificación poligonal, cuando se usa para la generación de los niveles de detalle, es remover primitivas de una malla original para producir modelos más simples los cuales retienen las características visuales más importantes del objeto original.

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

En el presente capítulo se hace un análisis de las técnicas de manejo de los niveles de detalles, así como de los principales algoritmos y tendencias que se están empleando en el mundo para la selección de los niveles de detalle y la deformación de las mallas.

## 1.1 Técnicas de manejo de los niveles de detalles

Las tres técnicas o *frameworks* básicos para el manejo de los niveles de detalles son: LOD discreto, LOD continuo y LOD dependiente de la vista. [3]

### 1.1.1 LOD discreto

El LOD discreto es el tradicional y más conocido (llamado también **LOD basado en rangos**), consistente en crear múltiples versiones (usualmente 3) para cada objeto durante un proceso *offline* o de preprocesamiento, cada una de ellas correspondiente a cada nivel de detalle, y en tiempo de ejecución se selecciona la versión adecuada.

Éste método tiene como ventaja que el proceso de simplificación puede demorar lo que sea necesario para generar los LOD; además, el hardware gráfico moderno permite la creación de versiones de LOD los cuales pueden ser compilados en preprocesamiento, adaptando las estructuras de los modelos a determinadas ventajas del hardware como las tirillas de triángulos (*triangle strips*), listas de visualización (*display lists*) o arreglos de vértices (*vertex arrays*).

Como esta es una operación precalculada, no se puede predecir desde qué dirección se estará viendo el objeto, por lo que este tipo de LOD se conoce también como isotrópico o independiente de la vista.

### 1.1.2 LOD continuo

Éste método, conocido también como **LOD progresivo**, es un tipo de LOD que parte de la idea del LOD discreto, con la diferencia de que el nivel de detalle de cada objeto es calculado en tiempo real cuando es necesitado, en lugar de ser preprocesado. El funcionamiento real consiste en preprocesar el objeto para crear una estructura asociada al mismo que contiene el espectro de detalles, la cual será utilizada en tiempo real para extraer los parámetros necesarios para el nivel deseado y modificar el objeto. [6]

De esta manera, nunca se tienen más polígonos de los necesarios, pues se tienen mejores aproximaciones a la cantidad de polígonos necesaria para cada nivel de detalle.

Con el LOD continuo se le da soporte además el *streaming* (fluido) de modelos poligonales, en la cual una base simple de un modelo es seguida por una serie de refinamientos que se van incorporando dinámicamente, lo cual es útil en la carga de grandes modelos desde disco o por la red, brindando un *rendering* progresivo y una carga ininterrumpible.

### 1.1.3 LOD dependiente de la vista

El LOD dependiente de la vista es una extensión del continuo, usándose un criterio de simplificación dependiente del punto de vista del observador, donde se selecciona dinámicamente el LOD más apropiado para la vista actual.

Como es un modelo anisotrópico, un simple objeto puede abarcar múltiples niveles de simplificación; por ejemplo, las partes más cercanas del objeto se pueden representar con mejor detalle que las más lejanas (óptimo para objetos grandes y complejos como terrenos), o la silueta se puede representar con mejor resolución que las regiones interiores.

### 1.1.4 LOD en la práctica

El LOD dependiente de la vista requiere un procesamiento extra en tiempo de ejecución que requiere evaluación, simplificación y refinamiento del modelo, por lo que debe usarse en situaciones que realmente lo requieran como en el *rendering* de terrenos.

El LOD continuo impone además un costo de procesamiento y de memoria, aunque se pudiera utilizar en algunos modelos en los juegos.

Por tanto, independientemente de las ventajas de los LOD continuo y dependiente de la vista, el LOD discreto tradicional es la mejor solución práctica, siendo el más simple y el que mejor trabaja sobre los *hardwares* gráficos más comunes.

### 1.1.5 Grafos de escena. Nodos de niveles de detalles

**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.

En las aplicaciones 3D en tiempo real se suele emplear una estructura jerárquica para la gestión de la escena denominada grafo de escena, el cual contiene información sobre la organización lógica y espacial de los objetos. El árbol que representa la escena está compuesto por una serie de nodos y se recorre en profundidad partiendo del nodo raíz hasta llegar a los nodos hojas realizando algún tipo de operación sobre estos nodos. El árbol se recorre de arriba abajo y de izquierda a derecha. [9]

Generalmente, un grafo de escena presenta nodos intermedios (que contienen información de los objetos como posición, orientación y escala, volúmenes fronteras (utilizados para el *culling* jerárquico y la detección de intersecciones), o cualquier otro estado geométrico o de *render* que se desee aplicar), y nodos hojas (objetos en cuestión). Al atravesar una rama y llegar a un nodo hoja, se tiene toda la información necesaria para el *rendering*. [4]

En general, el grafo de la escena facilita: [5]

- Segmentar en pequeñas piezas la escena (subárboles). Esto permite aplicarle efectos a una determinada parte de la escena independientemente, así como una eficiente transmisión de efectos.
- Localización espacial, útil en la eliminación de regiones, minimizándose la cantidad de datos enviados al *render*. Acelera además la detección de colisiones (“detección jerárquica”).

- Organización de los objetos en una jerarquía espacial y semánticamente, especialmente los caracteres humanoides.
- Salvar el estado del simulador con facilidad, salvándose cada nodo recursivamente.

**Nodos de niveles de detalles (LOD)**

Este es uno de los nodos típicos de un grafo de escena, y se corresponde con la implementación de los LOD discretos. [3]

Cuando el algoritmo de recorrido de la estructura jerárquica se encuentra con un nodo LOD, se efectúa una comprobación que escogerá un único hijo para ser recorrido y visualizado: aquel que corresponde al nivel de detalles óptimo según el estado geométrico del nodo. A través del grafo de la escena se pueden identificar para un mismo objeto sus LOD, por lo que durante el recorrido se puede escoger cuál de sus versiones representar. [9]

## 1.2 Selección del nivel de detalle

Los objetos y caracteres son representados en la gráfica computacional como modelos geométricos. Los modelos pueden ser creados a diferentes niveles de detalles (LOD), con más polígonos y mayores texturas para modelos de mayor detalle, y con menos polígonos y texturas pequeñas para modelos de menor detalle. [2]

Esto mejora el rendimiento del *renderer* y la calidad de la visualización ya que:

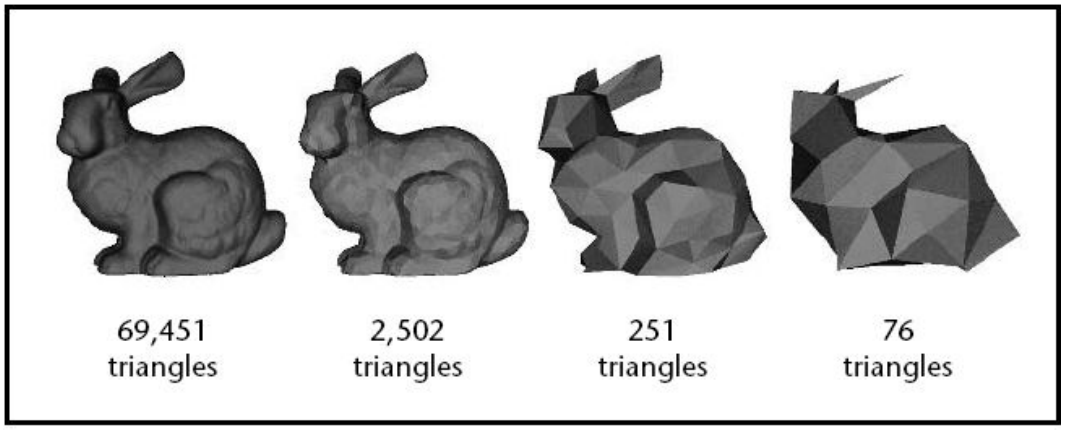
- Dibujar menos polígonos cuando los objetos están más lejos de la cámara aumenta la velocidad.
- Usando mayor detalle para los objetos que se encuentran cerca de la cámara se tiene una mayor calidad visual.

Para implementar el *rendering* por niveles de detalles, se crean múltiples modelos con diferentes niveles de detalles (tradicionalmente tres: cercano, medio y lejano), y se selecciona uno de ellos cada *frame* en dependencia de la distancia a la cámara. Por regla general, cada nivel suele tener el doble de polígonos que el nivel precedente a medida que se acerca a la cámara. [2]

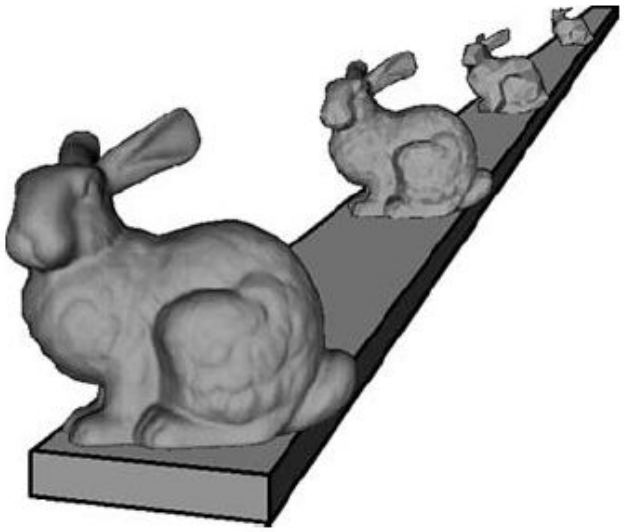
No todos los objetos requieren optimización por niveles de detalles, aquellos que permanecen a una distancia relativamente constante de la cámara (por ejemplo el personaje principal en un juego) se mantienen siempre a un mismo nivel de detalle. [2]

Existen situaciones en que es ideal el uso de la optimización por LOD, por ejemplo, cuando existe niebla en el escenario, donde a partir de cierta densidad y lejanía los objetos se representan con un mínimo de detalles. En el caso de los juegos y simuladores de vuelo, a partir del despegue se alejan todos los objetos en tierra pero a la vez es mayor el rango de la vista hasta el horizonte y por tanto los objetos a visualizar, por lo que se debe minimizar también el detalle del entorno a visualizar. [2]

La figura 4 representa el concepto fundamental del LOD: [3]




(a)



(b)

**Fig. 3 Modelos a diferentes niveles de detalles.**
**a) Modelos a diferentes LOD. b) Visualización de los modelos LOD**

### 1.2.1 Distancia a la cámara

La vía más simple de seleccionar el nivel de detalle es por la distancia a la cámara, por ejemplo, utilizando el detalle alto para modelos más cercanos que 500 unidades, el detalle medio para modelos entre 500 y 1500 unidades, y el detalle bajo para los modelos más lejanos que 1500 unidades de la cámara. Sin embargo, este método presenta dos problemas: [2]

Primeramente, no tiene en cuenta área que ocupa en pantalla. Si se hacen variaciones al lente de la cámara, provocará cambios en la visualización de los objetos: por ejemplo, si un objeto está lejos de la cámara pero el campo de visión o lente es muy estrecho (por ejemplo en un *zoom*), la imagen aparecerá grande en la pantalla y se necesitará un alto nivel de detalle; similarmente, aunque el objeto esté relativamente cerca de la cámara, si el lente es muy ancho el objeto parecerá pequeño y se necesitará menor nivel de detalle. La figura 5 ilustra el problema anterior.

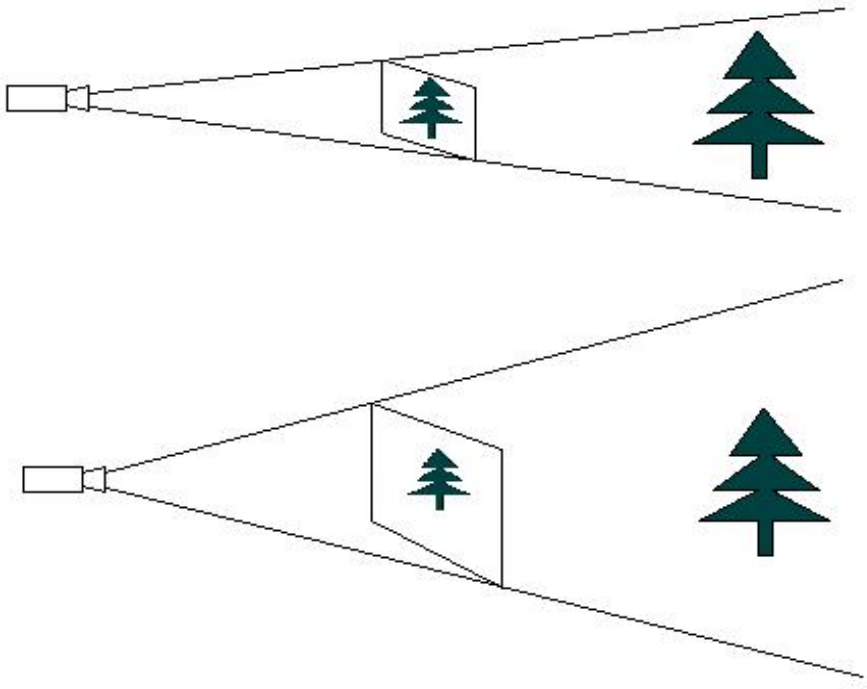


**Fig. 4 Variaciones de LOD en dependencia del campo de visión de la cámara**

Además, comúnmente existen objetos que a pesar de estar lejos de la cámara, son demasiado grandes como para eliminarles detalles, por ejemplo, un edificio que ocupa

un área importante de la pantalla; y a su vez, objetos que están cercanos pero son muy pequeños como para representarlos con mucho detalle, por ejemplo, una piedra pequeña; en estos casos tampoco es funcional ajustarles el detalle según la distancia a la cámara.

Una mejor alternativa a la distancia a la cámara, es el criterio de **área en pantalla**, el cual selecciona un mayor nivel de detalle a medida que el radio aumenta, como se explica en el siguiente epígrafe.

El segundo problema está dado cuando un objeto permanece cercano a las distancias límite, y está cambiando rápidamente de una profundidad a otra, por ejemplo, cuando un personaje esta corriendo en zigzag entre los niveles cercano y medio, alternando de niveles de detalles, provocando un efecto indeseable y de distracción. Este efecto se conoce como "*back and forth popping*", o "salto alante y atrás".

De manera general, cuando un objeto pasa de un nivel de detalle a otro, los cambios de polígonos (sobre todo en la silueta) y texturas no deben ser muy drásticos. Afortunadamente, ambos problemas pueden ser resueltos fácilmente (ver más adelante el epígrafe *"Hysteresis thresholding"*).

### 1.2.2 Área en pantalla. Volúmenes frontera

La selección basada en el área de pantalla [3] utiliza el objeto entero, siendo un criterio más realista de selección, pero a la vez, es más caro computacionalmente.

Muchos sistemas utilizan las cajas fronteras, proyectándolas en la pantalla y obteniendo una aproximación del área ocupada por el objeto. Esto presenta como inconvenientes que se trabaja con ocho puntos lo cual representa un gasto computacional.

Otra variante es transformar y proyectar el máximo y mínimo punto del objeto, en este caso de la caja frontera, y sustrayendo la posición en pantalla de cada coordenada para determinar el área en pantalla. Con éste método sólo se procesan dos puntos.

El inconveniente de ambos métodos radica en que dependen de la orientación del objeto. Una alternativa consiste en utilizar como volumen frontera una esfera, y calcular el radio proyectado por dicha esfera en la pantalla. Éste radio será el mismo independientemente de la orientación del objeto, pero es más pobre en cuanto al ajuste a la forma del objeto.

Existe la posibilidad de usar otros volúmenes frontera como elipsoides y cajas orientadas.

La figura 6 ilustra el efecto de usar cajas y esferas en la determinación del área ocupada en pantalla. a-b, cajas alineadas, dependientes de la orientación, c-d, esferas independientes de la orientación del objeto, pero menos ajustadas.

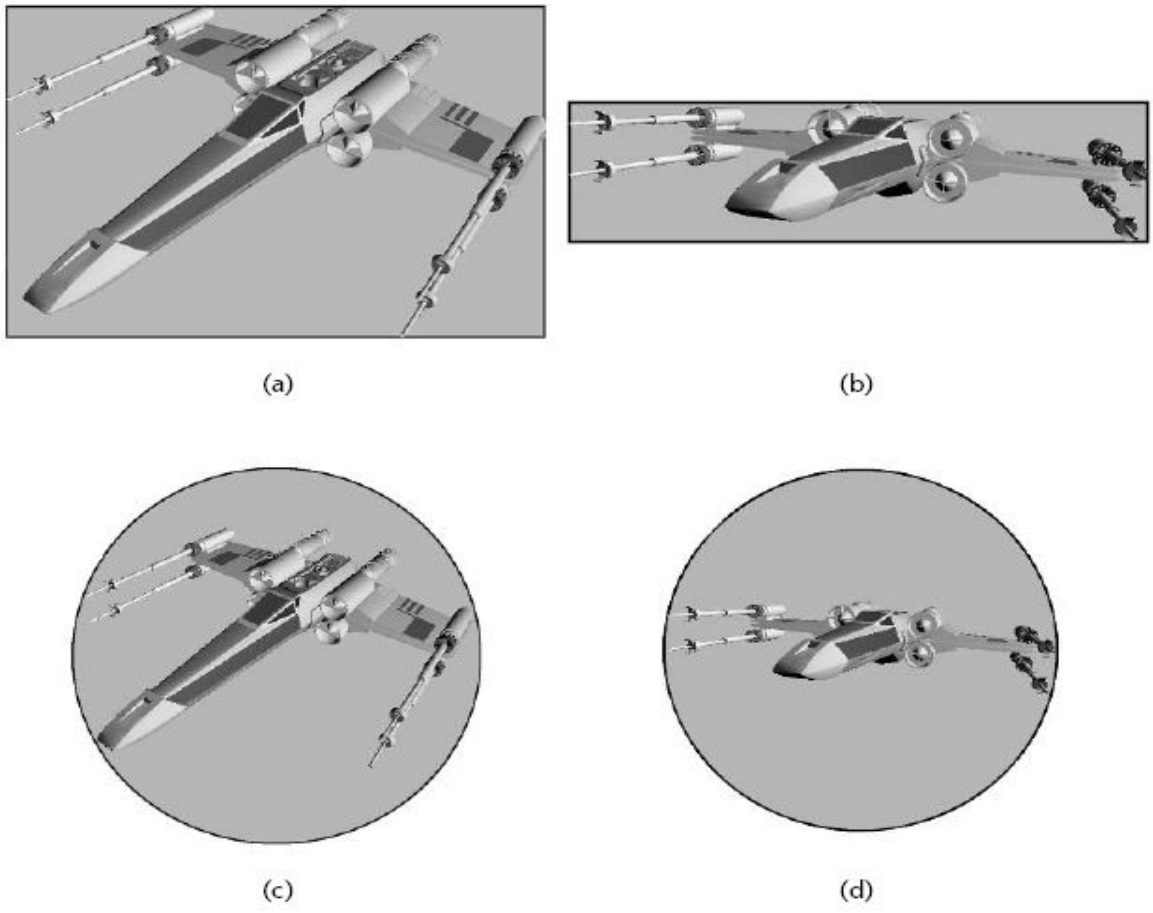



**Fig. 5 Uso de volúmenes frontera para determinar el área de proyección en pantalla del modelo**
**a-b, cajas alineadas, dependientes de la orientación.**
**c-d, esferas independientes de la orientación del objeto, pero menos ajustadas.**

### 1.2.3 Hysteresis thresholding

La "*hysteresis thresholding*" o histéresis de umbral [3], no es más que un retardo en la transición de niveles de detalles, en la cual los objetos pasan con mayor suavidad de un nivel a otro, con el objetivo de evitar el parpadeo ("*back and forth popping*") cuando un objeto constantemente salta entre dos niveles. Éste concepto se ilustra en la figura 7.

Se basa en el uso de un umbral superior e inferior construyéndose una franja de transición entre los límites de profundidades, haciéndose dentro de dicha franja un efecto de transición de un LOD a otro.

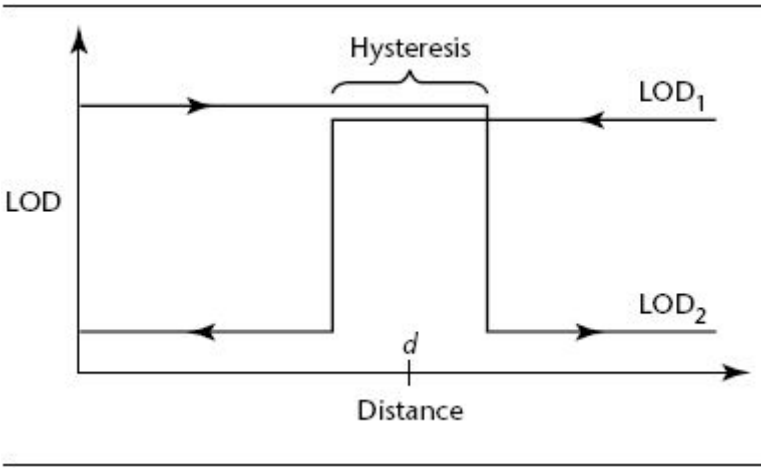



**Fig. 6 Histéresis alrededor de la distancia umbral "d"**

Una histéresis del 10% del rango de distancia del nivel de detalle brinda muy buenos resultados. Por ejemplo, considere la existencia de dos niveles de detalle LOD1 > LOD2 (figura 7), si la distancia de paso del LOD1 al LOD2 es de 100 metros, se puede escoger pasar del LOD1 al LOD2 a 110 metros, y pasar del LOD2 al LOD1 a 90 metros. La idea no es crear una transición durante el paso de un nivel a otro, sino lograr que si en un punto se cambió de detalle, que en ese mismo punto no se pueda regresar al detalle anterior. [3]

### 1.2.4 Otros factores de selección

Existen otros factores que se podrían tener en cuenta a la hora de seleccionar el nivel de detalle con que se va a representar un objeto [3], por ejemplo:

**La prioridad**: referida a que determinados objetos no deben seguir los criterios normales de selección, pues tienen una determinada importancia en la escena, y por tanto se les asocia niveles de prioridad altos o bajos que determinarán si se les aplica mayor o menor nivel de detalle.

**Condiciones ambientales**: factor referido anteriormente en este trabajo, y que se basa en que ante determinadas condiciones atmosféricas como niebla, lluvia, oscuridad o mucho brillo, el tratamiento de los objetos puede ser diferente a su tratamiento en condiciones normales.

**Factores perceptuales**: se basa en la idea de que el ojo humano percibe con menos detalle los objetos que están en la periferia, o aquellos que se mueven muy rápido frente a nuestra vista, por tanto, estos objetos podrían ser víctimas de un menor nivel de detalles.

# 1.3 Deformación de mallas

En el presente epígrafe se hace una referencia a los modelos de representación de objetos tridimensionales, así como a las topologías de las mallas poligonales a tener en cuenta en la deformación de las mallas.

## 1.3.1 Modelos de representación de objetos 3D

En los gráficos 3D se necesita disponer de una representación tridimensional de los elementos que formarán parte de la escena. Un objeto puede ser un punto, una línea, una superficie o un volumen.

Existen dos modelos para la visualización de objetos 3D: modelo de sólidos y modelo de superficies. [9]

Los **modelos de sólidos** se utilizan cuando se quiere representar la apariencia externa e interna de un objeto, con lo cual se necesita almacenar información sobre alguna magnitud que varía en el interior del objeto para poder representarla posteriormente. Estos modelos se utilizan mucho para almacenar imágenes médicas 3D, y para realizar determinadas operaciones entre objetos, tales como uniones o intersecciones, o aplicar ciertos algoritmos.

En los **modelos de superficies** solamente se representa información sobre la superficie y no del interior de los cuerpos pues no serán visualizados por dentro. Además los modelos de superficies son más adecuados para trabajar en tiempo real.

Para la aplicación de LOD a un modelo de superficie se necesita la siguiente información:

**Geometría del objeto**: forma y posición.

**Topología del objeto**: La topología consiste en definir cómo se relacionan las partes que lo forman, por ejemplo, un cubo definido por polígonos.

Existen básicamente dos tipos de representación de superficies: superficies curvas y modelo poliédrico. [9]

Las **superficies curvas** permiten una mayor facilidad a la hora de realizar el modelado de los objetos, ya que la mayoría de los objetos presentan superficies continuas curvadas. En muchas ocasiones, para modelar un objeto complejo se utiliza una descomposición de la superficie en trozos o parches (*patches*) triangulares o cuadrangulares, cada uno de los cuales se suele modelar son superficies paramétricas (Superficies de Bezier, NURBS). Entre estos tipos de representación se tienen las superficies implícitas (cuádricas, isosuperficies y superficies equipotenciales) y las superficies paramétricas (superficies de control).

La representación por **modelo poliédrico** consiste en definir los objetos a través de una superficie formada por polígonos que comparten sus aristas y vértices, es decir, de un poliedro que puede ser abierto o cerrado. Este modelo ofrece la ventaja de que es una representación más adecuada para la visualización utilizando hardware gráfico que trabaja con polígonos. El modelo poliédrico es rápido de visualizar puesto que sólo requiere operaciones lineales muy eficientes tanto para la proyección en la pantalla como para el posteriormente rellenado de los polígonos.

La principal desventaja de este método es que no siempre es fácil modelar con polígonos cualquier objeto, ya que la mayoría de los objetos no son realmente poliedros, sino que están compuestos por superficies continuas curvadas. Lo que normalmente se hace es realizar una aproximación de la superficie con el nivel de detalle que se considere oportuno.

### 1.3.2 Mallas poligonales. Topologías

Los conceptos esenciales de los LOD se aplican a cualquier representación de modelos. Tanto a las mallas poligonales, *splines*, *voxels*, superficies implícitas, e incluso representaciones basadas en puntos e imágenes, se les puede aplicar la optimización por LOD discreto, continuo y dependiente de la vista. Pero el uso más común e importante de los LOD sigue siendo la simplificación de mallas poligonales. [3]

Los modelos poligonales dominan actualmente los gráficos interactivos 3D por computadora (videojuegos, visualizaciones médicas y científicas, y CAD entre otros) por su simplicidad matemática. Debido a su representación de los objetos a través de piezas lineares o polígonos, permite la aplicación de algoritmos de representación regulares y simples, en los cuales la visibilidad y los colores de los píxeles son determinados interpolando a través de la superficie del polígono. [3]

Además, los polígonos sirven como denominador común para otros modelos de computadora, ya que la mayoría de las representaciones de modelos (*spline*, superficie implícita, superficies volumétricas) se pueden convertir con absoluta exactitud a mallas poligonales. Es por esto que muchísimas de las investigaciones de los LOD fijan su atención en el problema específico de simplificar mallas poligonales. [3]

Se puede definir un modelo poligonal como una colección de vértices y de polígonos que conectan a esos vértices, siendo los más conocidos la cinta de triángulos (*triangle strip*), el abanico de triángulos (*triangle fan*) y la malla de cuadriláteros (*quadrilateral mesh*), ver figura 8. [4]


**Fig. 7 Modelos poligonales**

La mayoría de los algoritmos de LOD simplifican realmente el problema si se asume que los acoplamientos poligonales han sido completamente triangulados, dado que la representación por triángulos mejora las prestaciones del *hardware*. [3]

La mayoría de las tarjetas gráficas visualizan más triángulos por segundo de los que pueden enviarse por el bus, es por esto que se plantean soluciones para minimizar la cantidad de triángulos y la información por vértice (compresión geométrica) como reducir el número de vértices (uso de conectividad, *strips*), reutilización de vértices (buffer de vértices) y reducción del número de polígonos (LOD). [3]

**Topologías** [3]

El tratamiento de la topología de mallas durante la simplificación, proporciona una importante distinción entre algoritmos. La topología de mallas es un asunto matemático formal y riguroso, por lo que a continuación se abordará el tema en un lenguaje informal evitando el uso de notaciones complejas.

En el contexto de la simplificación poligonal, la **topología** se refiere a la estructura en que están conectados los polígonos de las mallas.

El **género** es el número de agujeros en la superficie de la malla. Por ejemplo, una esfera y un cubo tienen género cero, mientras que un *toroide* y una taza de café tienen género uno.

La **topología local** en una cara, arista o vértice, se refiere a la conectividad con sus homólogos vecinos.

Una malla forma un 2D múltiple (*manifold*) si la topología local es equivalente a un disco, es decir, si la vecindad de cada característica (cara, aristas, o vértice) forma un anillo de polígonos conectados en una sola superficie. En una malla triangular con topología 2D múltiple, cada borde es compartido exactamente por dos triángulos, y cada triángulo comparte cada una de sus aristas con exactamente tres triángulos vecinos.

En una malla 2D múltiple con fronteras se puede identificar las aristas límites, donde cada una de ellas pertenece a solamente un triángulo (figura 9).

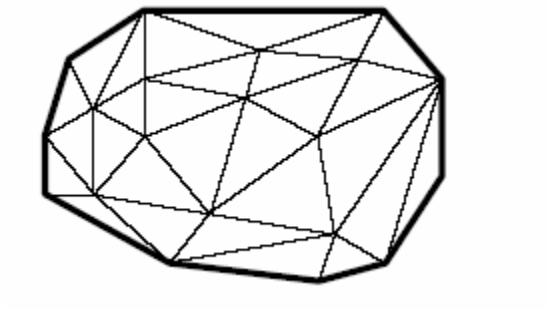


**Fig. 8 Malla 2D múltiple con límite (el límite en negrita)**

Las mallas múltiples son también deseables, por ejemplo, en el trabajo con elementos finitos en la ingeniería y simulaciones científicas, y en los algoritmos de iluminación y sombras por radiosidad.

Desafortunadamente, muchos modelos encontrados en la práctica real no son perfectamente múltiples, pues presentan defectos topológicos como grietas, empalmes en T, y puntos o aristas no múltiples (figura 10). Estos defectos son particularmente comunes en modelos hechos a mano, tales como los creados por artistas de videojuegos e ingenieros en sistemas CAD.

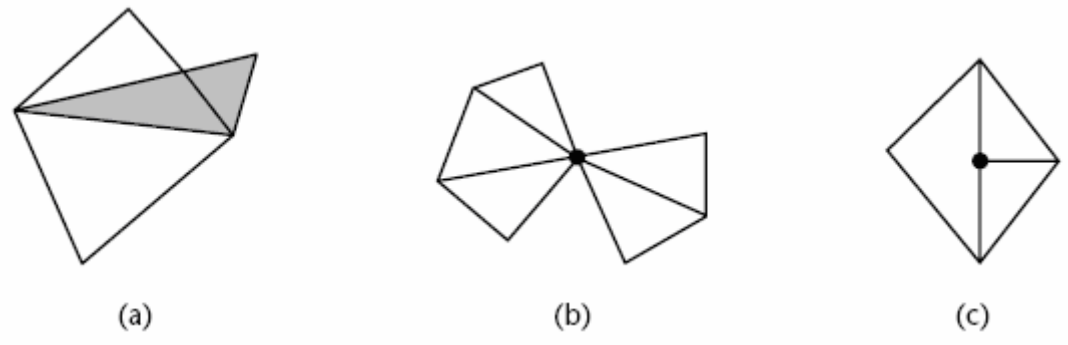



**Fig. 9 Ejemplos de acoplamientos no múltiples**
**(a) Una arista compartida por tres triángulos.**
**(b) Un vértice compartido por dos grupos desconectados de triángulos.**
**(c) Empalme en T: la arista de un triángulo es atravesada por aristas de otros dos triángulos.**

Virtualmente, cualquier algoritmo de simplificación puede funcionar con éxito en un objeto múltiple.

Los algoritmos que conservan la topología, preservan la conectividad múltiple en cada paso. Tales algoritmos no cierran los agujeros de las mallas, y por lo tanto preservan el género total. Puesto que no hay agujeros que aparecen o desaparecen durante la simplificación, la fidelidad visual del objeto simplificado tiende a ser relativamente buena.

Sin embargo, esta restricción limita las posibilidades de simplificación puesto que los objetos de alto género no pueden ser simplificados debajo de cierto número de polígonos sin que se cierren los agujeros del modelo (figura 11).

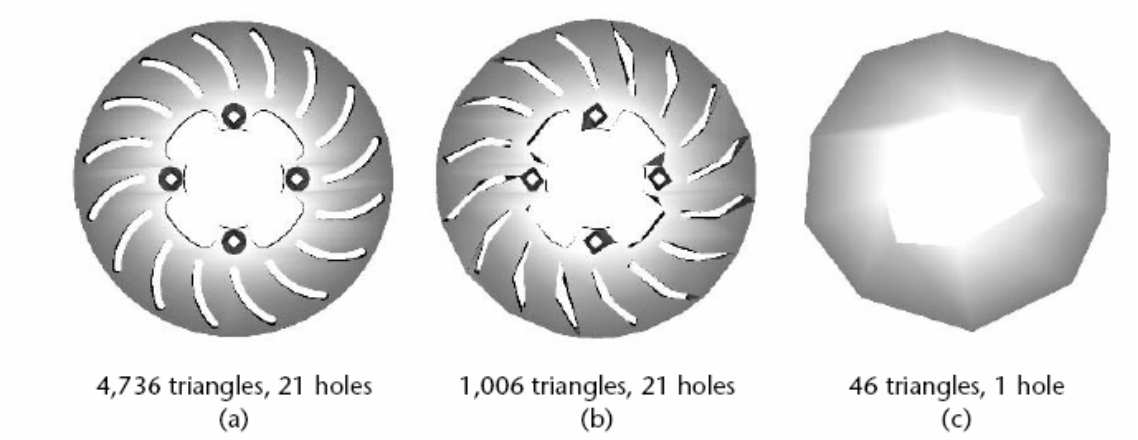



**Fig. 10 Preservación del género en la simplificación**
**(a) Modelo original de un rotor de freno**
**(b) Modelo simplificado conservando la topología, se limita la simplificación drástica**
**(c) Modelo simplificado sin conservar la topología.**

Un algoritmo conservador de la topología debe asumir que la malla inicial tiene topología múltiple, y no puede eliminar o combinar objetos pequeños, puesto que ambas operaciones violarían la propiedad de conectividad múltiple.

Algunos algoritmos son “tolerantes a la topología” e ignoran regiones en la malla con topología local no múltiple, dejando esas regiones sin simplificar. Otros simplemente fallan en estas regiones.

Los algoritmos modificadores de la topología no preservan necesariamente la topología múltiple. Pueden, por consiguiente, cerrar los agujeros en el modelo y ensamblar objetos separados, permitiendo simplificaciones más profundas que en los esquemas de conservación de la topología.

Esta simplificación drástica puede traer una fidelidad visual pobre, provocando distracción debido al parpadeo provocado por la aparición y desaparición de agujeros en el paso de un LOD al siguiente.

La mayoría de los algoritmos modificadores de la topología no requieren una topología válida en la malla inicial, lo cual aumenta grandemente su utilidad en aplicaciones CAD. Algunos algoritmos modificadores procuran regular el cambio en la topología, pero muchos son “insensibles a la topología” y no prestan ninguna atención a la conectividad en la malla inicial.

En general, los algoritmos conservadores de la topología son útiles cuando la fidelidad visual es crucial, o con aplicaciones tales como análisis con elementos finitos, en el cual la topología superficial puede afectar los resultados. La conservación de la topología también simplifica algunas aplicaciones, tales como la edición de superficies de multiresolución, donde se requiere una correspondencia entre las representaciones de alto y bajo detalle de un objeto.

La visualización en tiempo real de escenas muy complejas, sin embargo, requiere de simplificación drástica, y es aquí donde juegan su papel los algoritmos modificadores de la topología.

### 1.3.3 Deformación de mallas

Una malla en gráficos 3D tiene dos componentes: la geometría de la malla, representada por los vértices; y la conectividad de la malla, representada por los bordes y caras que conectan a los vértices. Esta conectividad codifica la topología de la malla, es decir, el número de huecos, túneles y cavidades en la malla. La simplificación de la geometría de la malla puede conllevar o no a la simplificación de la topología de la malla. [3]

A continuación se describe la simplificación de mallas como un proceso de optimización restringido por la "fidelidad" o por el "costo", a tener en cuenta en la aplicación de operadores locales y globales de simplificación. Los operadores **locales** simplifican la geometría y las conectividades en una región local de la malla reduciendo los polígonos, mientras que los operadores **globales** operan sobre regiones mayores y simplifican la topología de la malla. [3]

En la **simplificación basada en la fidelidad (*fidelity-based simplification*)**, se cota la simplificación de manera que satisfaga la malla original, minimizando el número de triángulos simplificados. Estas simplificaciones son convenientes para las aplicaciones en que la fidelidad visual es más importante que la interactividad.

Las restricciones de fidelidad son usualmente especificadas por la medida de las diferencias con la malla original, denotadas como error ε. Se pueden generar sucesivas simplificaciones, por ejemplo, con errores ε, 2ε, 4ε, 8ε, etc. Las técnicas de manejo del LOD pueden ser usadas para evaluar el error de permisible para un objeto en tiempo de ejecución, y determinar el nivel de detalle apropiado. [3]

En la **simplificación basada en el costo (*budget-based simplification*)**, el usuario especifica el número máximo de triángulos, y el algoritmo trata de minimizar el error ε evitando exceder éste límite de triángulos. A pesar de que este algoritmo genera un número fijo de triángulos, es apropiado para aplicaciones críticas donde es importante una frecuencia de dibujado deseada, principalmente en aplicaciones donde la interactividad es más importante.

Como el error ε no es controlable por el usuario final, esta solución no garantiza la fidelidad visual, y los algoritmos de este tipo frecuentemente se basan en observaciones empíricas de grupos de datos. [3]

A continuación se presentan varios operadores locales utilizados para la simplificación de mallas (colapsamiento de aristas, colapsamiento de aristas virtuales (pares de vértices), colapsamiento de triángulos, colapsamiento de celdas, eliminación de vértices, combinación de polígono y sustitución geométrica general).

### 1.3.3.1 Colapsamiento de aristas

Este operador [3] colapsa una arista $(v_a, v_b)$ a un solo vértice $v_{New}$, causando la eliminación de la arista $(v_a, v_b)$ y de los triángulos que contienen a esa arista. El operador inverso de un colapsamiento de arista es la partición de vértices, que agrega la arista $(v_a, v_b)$ y los triángulos adyacentes a ella. Así, el operador de colapsamiento de arista simplifica la malla y el operador partición de vértices agrega detalle a la malla. La figura 12 ilustra el colapsamiento de arista y su contrario, la partición de vértices.

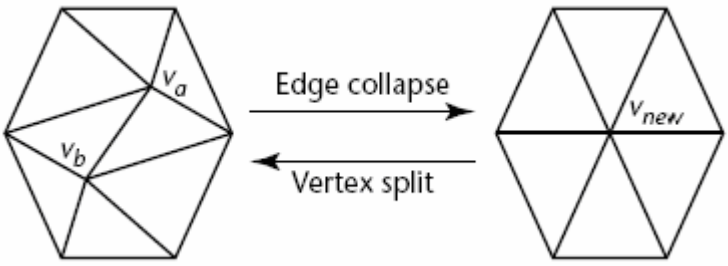



**Fig. 11 Colapsamiento de aristas y su inverso, partición de vértices.**

Existen dos variantes del operador de colapsamiento de arista: el colapsamiento de media arista y el colapsamiento de arista completa. En el colapsamiento de media arista (figura 13), el vértice a el cual se le colapsa la arista es uno de sus puntos finales, es decir, $v_{New} = v_a$ ó $v_{New} = v_b$. En el colapsamiento de arista completa (más general, abreviado a menudo como simplemente colapsamiento de arista) el vértice colapsado $v_{New}$ puede ser un vértice nuevamente computado.

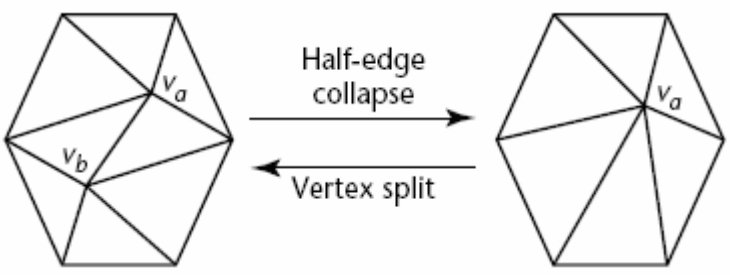



**Fig. 12 Colapsamiento de media arista y partición de vértice.**

Aunque el operador de colapsamiento de arista es simple de implementar, se debe tener cuidado de no aplicarlo si este causaría un pliegue de la malla o una inconsistencia topológica.

Los **pliegues en la malla** son un efecto secundario indeseado de algunos colapsamientos de aristas. En la figura 14, cuando la arista *($v_a$, $v_b$)* del triángulo *($v_d$, $v_c$, $v_a$)* se colapsa al nuevo vértice $v_{New}$, la malla alrededor del vértice $v_{New}$ "se dobla" debido al triángulo recientemente creado *($v_d$, $v_c$, $v_{New}$)*. Esto puede ser detectado midiendo el cambio en las normales de los triángulos correspondientes, antes y después de colapsamiento de arista: un pliegue en la malla se caracteriza por un cambio grande en el ángulo de la normal, generalmente mayor de 90°. Los pliegues en la malla traen como resultado efectos visuales indeseables tales como discontinuidades de la iluminación y de la textura, donde no existía ninguno antes.

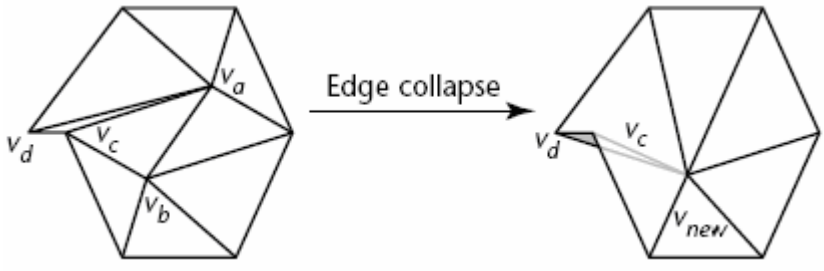



**Fig. 13 Pliegue de la malla presente en un colapsamiento de arista.**

En el siguiente ejemplo (figura 15) se puede observar una **inconsistencia topológica**: si las vecindades de los vértices *va* y *vb* comparten tres o más vértices, el colapsamiento de la arista *($v_a$, $v_b$)* creará una o más aristas no múltiples (aristas con uno, tres, o más triángulos adyacentes). Puesto que muchos algoritmos dependen de la conectividad múltiple, introducir tales aristas puede crear problemas más adelante en el proceso de simplificación.

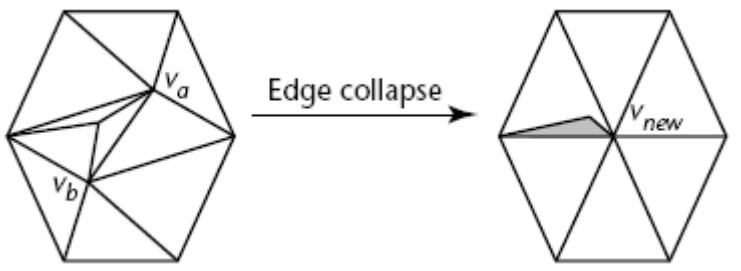



**Fig. 14 Conversión de una malla múltiple a no múltiple durante un colapsamiento de arista.**

El colapsamiento de aristas se ha utilizado ampliamente en la simplificación independiente de la vista, simplificación dependiente de la vista, compresión progresiva, así como la transmisión progresiva.

### 1.3.3.2 Colapsamiento de aristas virtuales (pares de vértices)

El operador de colapsamiento de pares de vértices [3], colapsa dos vértices inconexos $v_a$ y $v_b$, como muestra la figura 16.

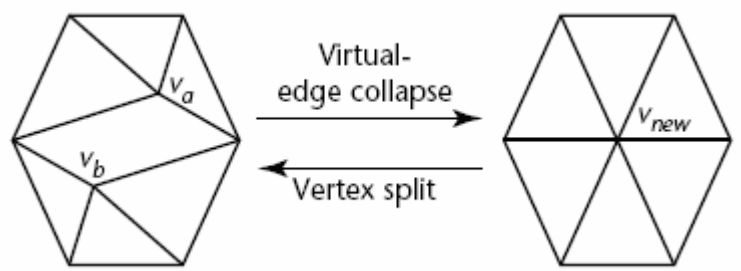



**Fig. 15 Colapsamiento de arista virtual o par de vértices**

Puesto que estos vértices no comparten una arista, no se destruye ningún triángulo, sino que los triángulos que rodean a $v_a$ y $v_b$ se actualizan como si una arista imaginaria que conecta a $v_a$ y $v_b$ experimentara un colapsamiento de arista. Por esta razón, el operador de colapsamiento de pares de vértices ha sido también llamado **colapsamiento de arista virtual**. Colapsar vértices no conectados permite la conexión de componentes no relacionados, así como el cierre de agujeros y de túneles.

En general, para una malla con $n$ vértices puede haber potencialmente $O(n^2)$ aristas virtuales: un algoritmo que considere todas las aristas virtuales posibles funcionará lentamente. La mayoría de los algoritmos de simplificación de topología que realizan el colapsamiento de aristas virtuales utilizan algún heurístico para limitar las aristas virtuales candidatas a un número pequeño. Uno de estos heurísticos, por ejemplo, elige solamente las aristas virtuales formadas entre los vértices próximos (con una distancia δ pequeña) a un vértice $v_a$. En la práctica esto reduce el número de las aristas candidatos a un factor linear del tamaño modelo, aunque se debe tener cuidado de elegir un δ bastante pequeño.

### 1.3.3.3 Colapsamiento de triángulos

Un operador de colapsamiento de triángulos [3] simplifica una malla colapsando un triángulo $(v_a, v_b, v_c)$ a un solo vértice $v_{New}$. Las aristas que definen la vecindad del $v_{New}$ son la unión de las aristas de los vértices $v_a$, $v_b$, y $v_c$. El vértice $v_{New}$ puede ser uno de los vértices $v_a$, $v_b$, o $v_c$, o un vértice nuevo computado (figura 17).

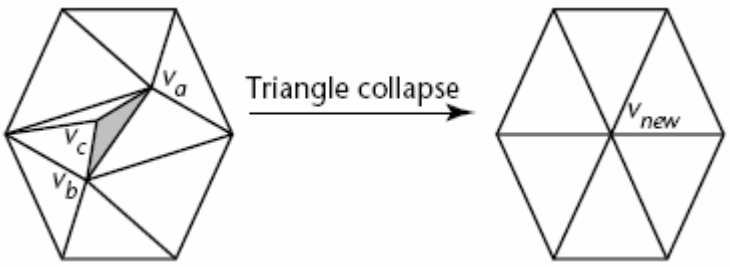



**Fig. 16 Colapsamiento de triángulos.**

Un colapsamiento de triángulo es equivalente a dos colapsamientos de aristas. Una jerarquía basada en colapsamiento de triángulo es menos profunda que una jerarquía basada en colapsamiento de aristas por lo que requiere de menos memoria. Sin embargo, las jerarquías de colapsamiento de triángulo son también menos adaptables, puesto que el colapsamiento de triángulo es una operación menos fina que un colapsamiento de arista.

### 1.3.3.4 Colapsamiento de celdas

El operador de colapsamiento de celda [3] simplifica la malla de entrada colapsando todos los vértices en cierto volumen o celda, a un solo vértice. Esta celda podría pertenecer a una rejilla o a una subdivisión espacial tal como un *octree*, o se podría definir simplemente como un volumen en el espacio. El vértice simple al cual las celdas se colapsan se podría elegir a partir de uno de los vértices colapsados, o computado nuevamente como un promedio de los vértices colapsados.

Considere un objeto de malla triangular como se muestra en la figura 18 (a). Los vértices de la malla se ponen en una rejilla regular figura 18 (b), y todos los vértices que caen en la misma celda se unifican en un solo vértice. Todos los triángulos de la malla original que tienen dos o tres de sus vértices en una sola celda se simplifican a una sola arista o a un solo vértice (figura 18 (c)). La malla final se muestra en la figura 18 (d). Note que tal simplificación no preserva la topología de la malla de entrada, y que el nivel de la simplificación depende de la resolución de la rejilla.

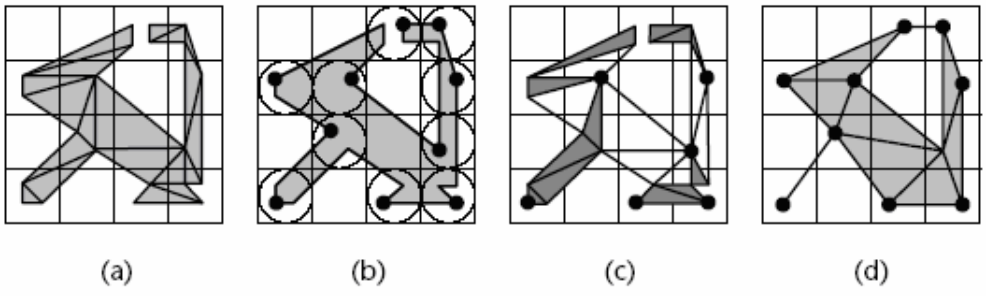



**Fig. 17 Colapsamiento de celdas.**

**(a) Una rejilla regular clasifica los vértices. (b) Un solo vértice se selecciona para representar todos los vértices dentro de cada celda. (c) Los triángulos con 2 ó 3 vértices de la esquina en la misma celda simplifican a una sola arista o vértice, respectivamente. (d) La simplificación final.**

### 1.3.3.5 Eliminación de vértices

Este operador [3] quita un vértice junto con sus aristas y triángulos adyacentes, y triangula el agujero resultante (figura 19).

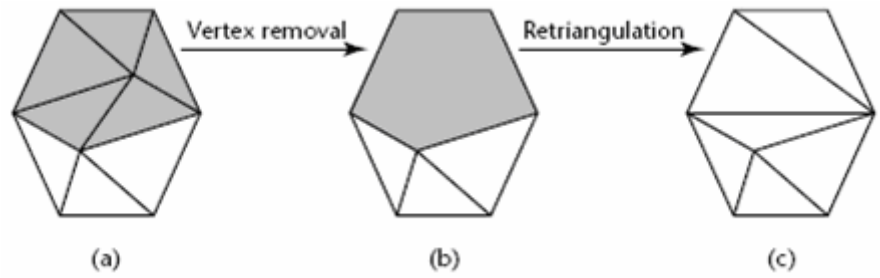



**Fig. 18 Eliminación de vértices**

**(a) Un vértice es escogido para ser eliminado. (b) El vértice y los triángulos adyacentes son eliminados, dejando un hueco en la malla. (c) El hueco es retriangulado con dos triángulos menos que la malla original.**

La triangulación del agujero en la malla resultante al retirar el vértice *v*, se puede lograr de varias maneras y una de estas triangulaciones es igual a un colapsamiento de media arista que implica al vértice *v*. La eliminación del vértice puede ser considerado una generalización del operador de colapsamiento de media arista.

### 1.3.3.6 Combinación de polígonos

Este operador [3] combina polígonos casi coplanarios y adyacentes en polígonos más grandes, los cuales son entonces triangulados. Como ejemplo, considere los triángulos sombreados en ella figura 19(a) que han sido combinados en (b) y después retriangulados en (c). La combinación de polígonos es más general que la eliminación de vértices puesto que puede combinar los polígonos (triángulos no exactos), dar lugar al retiro de varios vértices inmediatamente, e incluso obtener polígonos combinados con los agujeros.

La combinación de polígonos se ha utilizado en diferentes aplicaciones como los *face clustering*. La figura 20 ilustra un ejemplo.

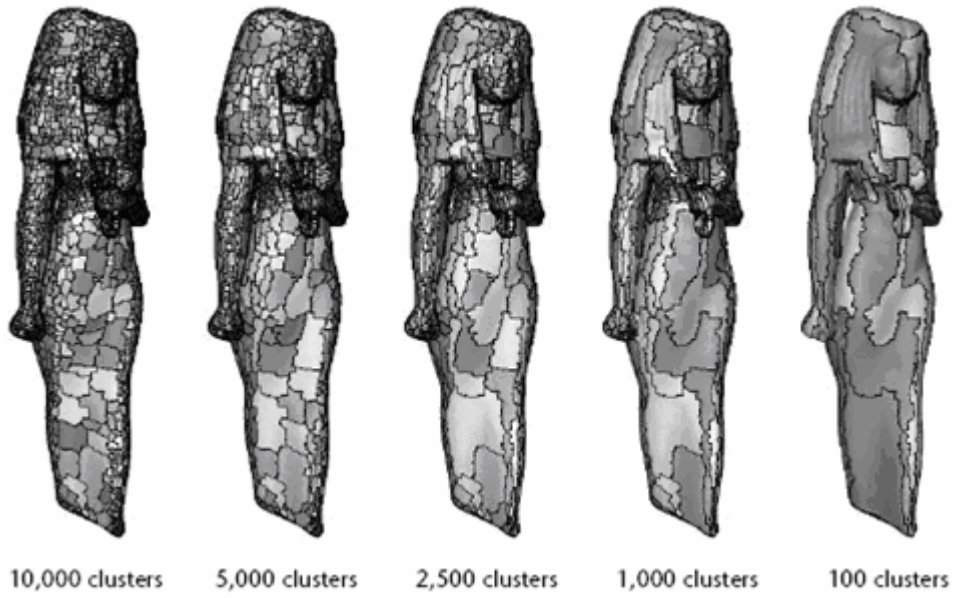



**Fig. 19 Face clustering**

### 1.3.3.7 Sustitución geométrica general

El operador de sustitución geométrica general (llamado también de **multitriangulación**), [3] sustituye un subconjunto de triángulos adyacentes por otro conjunto de triángulos simplificados, tal que sus límites sean iguales.

Existe el criterio de que este operador es el más general de los operadores de simplificación de malla. Puede incluso ser utilizado para sustituir geometrías primitivas de un tipo a otro, por ejemplo, para sustituir primitivas de punto por primitivas de triángulos.

La sustitución geométrica general puede codificar colapsamientos de aristas y eliminación de vértices. Además codifica el volteo de aristas (*edge flip*), donde la arista común entre dos triángulos es sustituida por otra arista que une los otros dos vértices opuestos. Se ha demostrado que el colapsamiento de aristas y el *edge flip* son suficientes para generar cualquier simplificación de geometría de malla y por lo tanto el operador de sustitución geométrica general es suficiente para codificar todas las simplificaciones en la geometría de malla.

### 1.3.3.8 Comparación de operadores locales de simplificación

Los operadores de colapsamiento (colapsamiento de aristas, de triángulos, y de celda) son los más simples de poner en ejecución. Además, puesto que pueden ser conceptualizados como una reducción gradual de la primitiva geométrica apropiada (arista, triángulo, o celda, respectivamente) a un solo vértice, son buenos para implementar transformaciones entre sucesivos niveles del detalle. [3]

El operador de colapsamiento de arista completa tiene mayor flexibilidad para determinar el nuevo vértice en comparación con el de media arista, permitiendo computar simplificaciones de mayor fidelidad. [3]

Sin embargo, la ventaja del operador de colapsamiento de media arista es que los vértices de la malla simplificada son un subconjunto de la malla de entrada, por lo que no se almacena ningún vértice nuevo. Finalmente, el número de triángulos modificados por un colapsamiento de media arista es más pequeño que el número modificado por un colapsamiento de arista completa. Esto puede conducir a una simplificación más eficiente puesto que pocos triángulos necesitan ser actualizados. [3]

El operador de colapsamiento de celda tiene muchas ventajas: proporciona geometría y simplificación topológica, es robusto, y es muy simple de poner en ejecución. También tiene desventajas: primero, es variable con respecto a la rotación y la traslación, esto significa que un objeto se puede simplificar de diferentes formas dependiendo de su rotación y posición, puesto que diversas partes del objeto terminarán arriba en diversas celdas de la rejilla. En segundo lugar, el colapsamiento de celda no simplifica la topología de una manera controlada. Por ejemplo, un agujero pequeño que atraviesa un límite de celda se puede agrandar, pero si el mismo agujero cae completamente dentro de una celda, desaparecerá. A pesar de estas desventajas el operador colapsamiento de celda tiene mucho para ser recomendado. [3]

El operador de eliminación de vértices requiere más esfuerzos que los operadores de colapsamiento para ser implementado correctamente, puesto que implica la triangulación de un agujero en 3D. Hay varios algoritmos públicos disponibles de LOD basados en el operador de eliminación de vértices, tal como el *Visualization ToolKit* o *VTK* que pueden servir como punto de partida para quien se inicia. [3]

Los operadores de eliminación de vértices y de colapsamiento de aristas tienen sus ventajas y desventajas, pero ninguno es estrictamente más poderoso que el otro. Cada uno produce mallas triangulares que no puede producir el otro. Computar el nuevo vértice de salida de un colapsamiento de arista es un problema de optimización continuo, mientras que elegir entre las diferentes maneras posibles de llenar el agujero después de un retiro de vértice es un problema de optimización discreto. El número de triángulos afectados por la eliminación de vértices es más pequeño que el que se afecta con el colapsamiento de arista (es decir, los triángulos circundantes a un vértice en oposición a dos) [3]

Es interesante observar que hay un subconjunto de las operaciones posibles del colapsamiento de aristas que son equivalentes a un subconjunto de las operaciones posibles de la eliminación de vértices. Este subconjunto común es la entrada de los colapsamientos de media arista, que comparten algunas de las características de cada uno de los otros dos tipos operaciones. [3]

Por último, la puesta en práctica del operador de sustitución geométrica general requiere un poco más de esfuerzo para capturar su generalidad completa, pero una vez implementado ofrece mayor flexibilidad. [3]

Existen otros operadores [7], siendo básicamente los siguientes:

**Normalización:** eliminación de caras degeneradas o aristas y cualquier primitiva definida múltiples veces.

**Simplificación de bordes:** eliminación de todos los bordes más pequeños que algún umbral.

**Simplificación basada en ángulo:** eliminación de aristas que forman un ángulo cerrado. Inversamente, los bordes que están alineados son combinados.

**Simplificación del tamaño de la cara:** eliminación de todas las caras con un área más pequeña que algún umbral. Los hoyos pudieran tener que ser llenados.

**Simplificación de la normal de la cara:** combinación de todas las caras adyacentes con normales casi paralelas.

### 1.3.3.9 Operadores globales de simplificación

Los operadores globales de simplificación [3] modifican la topología de la malla, y tienden a ser más complejos que los operadores locales de simplificación, que consideran solamente una pequeña porción del modelo. Incluyen métodos basados en volumen, procesamiento digital de señales, tetrahedralización de Delaunay, revestimiento de alfa, y otros métodos de filtración recientes basados en la teoría de Morse.

Se han propuesto dos esquemas para la simplificación de la topología en el dominio volumétrico: filtración *low-pass* y operaciones morfológicas de dilatación y erosión. Ambos esquemas convierten los objetos de entrada a una rejilla volumétrica, aplican operaciones de simplificación de la topología, y finalmente, con un método de extracción isosuperficie, convierten las densidades volumétricas en una malla triangular.

La filtración *low-pass* elimina los detalles finos en el modelo volumétrico, incluyendo características topológicas tales como agujeros y túneles pequeños. Aplicando al objeto original un filtro *low-pass* de un soporte apropiado, se crean niveles de jerarquía de simplificación, los cuales son usados alternadamente para crear un LOD poligonal.

El esquema de operadores morfológicos volumetriza el modelo de entrada en un juego de datos volumétrico usando un esquema de comprobación de paridad para un solo modelo poligonal componente. El operador de dilatación con un umbral T agranda el objeto eficazmente, llenando sus agujeros y conectando los componentes desconectados que quedan dentro de una distancia T. El operador de erosión es el complemento del operador de dilatación y hace una contracción del objeto. Un operador de dilatación seguido por un operador de erosión dará lugar a un objeto topológico simplificado con las mismas dimensiones que el original. Los operadores de dilatación y de erosión son muy exactos y pueden ser utilizados para controlar finalmente el nivel de simplificación topológica del objeto, preservando la forma del modelo.

Cuándo es deseable la simplificación de la topología [3]

- La simplificación de la topología es obligada y a menudo vital para la visualización interactiva de aplicaciones.

- Si las topologías se pueden simplificar en algunos casos a un solo agujero, se logrará una simplificación geométrica mucho más agresiva sin sacrificar el realismo visual.
- Las simplificaciones de la topología de los agujeros de píxeles secundarios pueden también ayudar a reducir los efectos de *aliasing*, que no es más que la apariencia punteada de las imágenes, que ocurre cuando la resolución no es suficiente o cuando las imágenes se han agrandado.

Sin embargo, hay varios usos donde la simplificación de la topología es indeseable y conduce a cambios inaceptables en la estructura del objeto. Por ejemplo en los modelos mecánicos CAD. El análisis finito de elementos para determinar la fuerza o la tensión en diversas partes de una parte mecánica puede requerir la presencia continua de ciertas estructuras topológicas tales como agujeros y vacíos. La simplificación topológica de tales estructuras puede a veces alterar drásticamente los resultados de tales simulaciones y debe por lo tanto ser evitada.

De manera semejante, en la proyección de imágenes médicas los datos recogidos por exploraciones automatizadas de tomografía (CT) o de la proyección de imágenes por resonancia magnética (MRI) tienen a menudo estructuras topológicas importantes que es mejor dejar en los datos, lejos de simplificarlas.

## 1.4 Niveles de detalles con imágenes

De forma complementaria a las técnicas multiresolución basadas en la geometría, existen las técnicas de visualización basadas en imágenes que modifican parámetros de las imágenes de los objetos logrando diferentes niveles de detalles, o sustituyen objetos completos por imágenes sencillas. A continuación se mencionan algunas de estas técnicas.

### 1.4.1 Uso de Impostores

Básicamente, el uso de **impostores** se refiere a la idea de sustituir una geometría compleja por una imagen 2D de aquello que debería estar representado. [8]

Desde sus primeras versiones OpenGL soporta el PTM, la proyección de una textura (*projective texture mapping*).

Las técnicas de representación de sólidos basándose en imágenes pueden considerarse dependientes del observador, ya que en función del punto de vista en un instante, se selecciona la imagen correspondiente. La complejidad del modelo geométrico sobre el que se aplican estas imágenes a modo de textura varía desde un simple plano hasta mallas más o menos complejas. [8]

La generación del conjunto de imágenes a utilizar como impostores se realiza una única vez por cada objeto a representar, y desde diversos puntos de vista. Se utiliza la proyección ortográfica (cámara ortogonal), para obtener una vista del objeto independiente de la distancia del plano de proyección a él, controlando que el objeto quepa totalmente en el plano de proyección. De esta manera por cada objeto a optimizar se pueden obtener imágenes para varios niveles de detalles, es decir, varias vistas del objeto. [8]

Generalmente, al almacenar estas imágenes se suelen controlar también la información acerca de la matriz de proyección y la posición de la cámara utilizada para cada toma. En las imágenes obtenidas, los píxeles pertenecientes al objeto tienen un valor alpha 0.0, mientras que los correspondientes al fondo, totalmente blancos, se almacenan con el valor alpha a 1.0, es decir son transparentes, permitiendo ver el resto de los objetos normalmente. [8]

### 1.4.2 Morphing

El *morphing* se refiere a una técnica de animación, en la cual una imagen es transformada gradualmente en otra por la distorsión de los puntos correspondientes. Se puede comenzar con dos imágenes (por ejemplo, un aeroplano y una libélula) y terminar con un insecto metálico. [3]

En películas y animaciones se usa muy a menudo para representar a una persona que se convierte en otra con algunos medios mágicos o tecnológicos, o como parte de una fantasía o secuencia surreal. [3]

En términos de niveles de detalles por multiresolución basada en imágenes, donde por diferentes niveles de detalles se sustituye el objeto en cuestión por imágenes de menor o mayor calidad, se puede utilizar el *morphing* para suavizar las transiciones bruscas que se producen al pasar de una imagen con un determinado nivel de resolución a otra con diferente nivel de resolución. [9]

## Conclusiones

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

De esta forma, se tienen los elementos necesarios para definir y caracterizar el módulo a desarrollar.

# 2

# Capítulo 2 Descripción de la solución propuesta

## Introducción

En este capítulo se brinda una visión del sistema a desarrollar, a través de una descripción técnica de la solución como resultado del estudio realizado en el capítulo anterior. Se inicia además la concepción práctica del producto a elaborar.

## 2.1 Soluciones técnicas

El sistema a implementar tendrá las siguientes características:

Se implementarán las técnicas de niveles de detalles discreto (con tres modelos por defecto), y continuo. Para esta versión no se podrán definir los modelos discretos por diseño, sino que se construirán los mismos en tiempo de preprocesamiento en un proceso semejante al del LOD continuo.

No se implementará ni se dará soporte al LOD dependiente de la vista.

En concordancia con las estructuras de datos de la biblioteca de la cual formará parte este módulo, se trabajará solamente con modelos de superficies poliédrico, específicamente las mallas triangulares. Además no se modificarán las clases existentes para el manejo de las geometrías, sino que se implementarán clases que tomen la información de estas geometrías y la modifiquen.

No se crearán nodos específicos para el manejo del LOD en el grafo de escena existente, sino que a los nodos que almacenan las geometrías se les podrá atachar las clases que modificarán la información de las geometrías que contengan dichos nodos. Básicamente, estos nodos contendrán la malla en sí, y podrán tener un modificador del detalle de la malla, el cual contendrá los parámetros y datos (por ejemplo, el espectro de modelos) necesarios para esta modificación.

La selección del detalle se hará por distancia a la cámara para esta versión, pero se dará soporte a una futura selección por área en pantalla. Para esto, cada modelo del espectro contendrá la distancia a la cámara a la cual es aplicable, y el área en pantalla para la cual es aplicable también, aunque solamente se calculen las distancias a la cámara en esta versión.

En el caso del LOD discreto se tendrá en cuenta la histéresis de umbral (*hysteresis thresholding*) en las zonas de cambio, con un valor por defecto del 10%. La misma se aplicará actualmente para la distancia a la cámara, pero se podrá aplicar de manera semejante al área en pantalla.

No se tendrán en cuenta otros factores de selección como las condiciones ambientales de niebla, lluvia, oscuridad o brillo, ni factores perceptuales del ojo.

Dentro de los parámetros a modificar estará solamente la malla, es decir se trabajará sobre su deformación, sin que esto incluya uso de impostores, *morphing*, o el cambio de calidad en las imágenes.

La malla se deformará con el operador local de colapsamiento de aristas (colapsamiento de media arista), conservándose la topología y género de la malla, y basado en la fidelidad (se introduce una métrica para minimizar los cambios drásticos entre modelos continuos del espectro) y el costo (se configura la menor cantidad de polígonos a la que se podrá deformar una malla).

Se programará en C++ estándar siguiendo la filosofía Orientada a Objeto acorde con el trabajo hecho en la biblioteca básica.

## 2.2 Reglas del negocio

Las reglas del negocio describen las restricciones explícitas de comportamiento del sistema, definiendo el límite entre actividad de negocio aceptable y no aceptable. Para este dominio de trabajo quedan definidas las siguientes reglas:

- La técnica LOD discreto usará 3 modelos.
- No se podrán definir los modelos discretos por diseño, sino que se generarán por software.
- Cada objeto al que se le desee modificar el detalle tendrá un único modificador, y cada modificador podrá ser asignado a un único objeto.
- Se definirá la cantidad de polígonos a partir de la cual a los objetos se les asociará modificadores de detalles, esto podrá ser utilizado para hacer esta acción automáticamente por ejemplo durante la carga de los objetos de la escena, de manera que se le aplique a todos los que tengan más polígonos que los especificados. En caso de que se desee asociar un modificador de detalles a un objeto que tenga menos polígonos, se deberá especificar por parámetro que se ignore esta comparación.
- Se tomarán los siguientes valores por defecto:
    - Límite para aplicar LOD: 2000 triángulos.
    - Máxima compresión que tendrán las mallas: 500 triángulos.
    - La zona cercana estará entre los valores 100.0f y 500.0f.
    - La zona media estará entre los valores 500.0f y 1500.0f.
    - La zona lejana estará entre los valores 1500.0f y 3000.0f.
    - Histéresis: 10%
- Se validará la lógica de los datos anteriores de manera que la histéresis no sobrepase el 100%, y la zona cercana tenga valores menores que la zona media, y la zona media tenga valores menores que la zona lejana. En caso de entrar datos incorrectos se establecerán los valores por defecto o se mantendrán los últimos datos correctos establecidos.
- Cuando un objeto esté más cerca que el límite inferior de la zona cercana, o más lejos que el límite superior de la zona lejana, se le mantendrá el detalle correspondiente a dichos límites respectivamente.

## 2.3 Modelo del dominio

La herramienta de desarrollo implementada en el proyecto (“Scene ToolKit”) está concebida como la representación computacional de un conjunto de módulos, es por esta razón que este módulo no representa un negocio en su totalidad y no se debe tratar como tal, sino como un dominio, conociendo qué información es relevante en el dominio y cómo es el entorno en el que se ubicará el módulo en cuestión.

Este modelo servirá como referencia a usuarios y programadores para la comprensión de todos los términos manejados en la elaboración del módulo, así como guía inicial para el futuro diseño.

Se representan a continuación los conceptos y sus relaciones en un diagrama de clases UML:

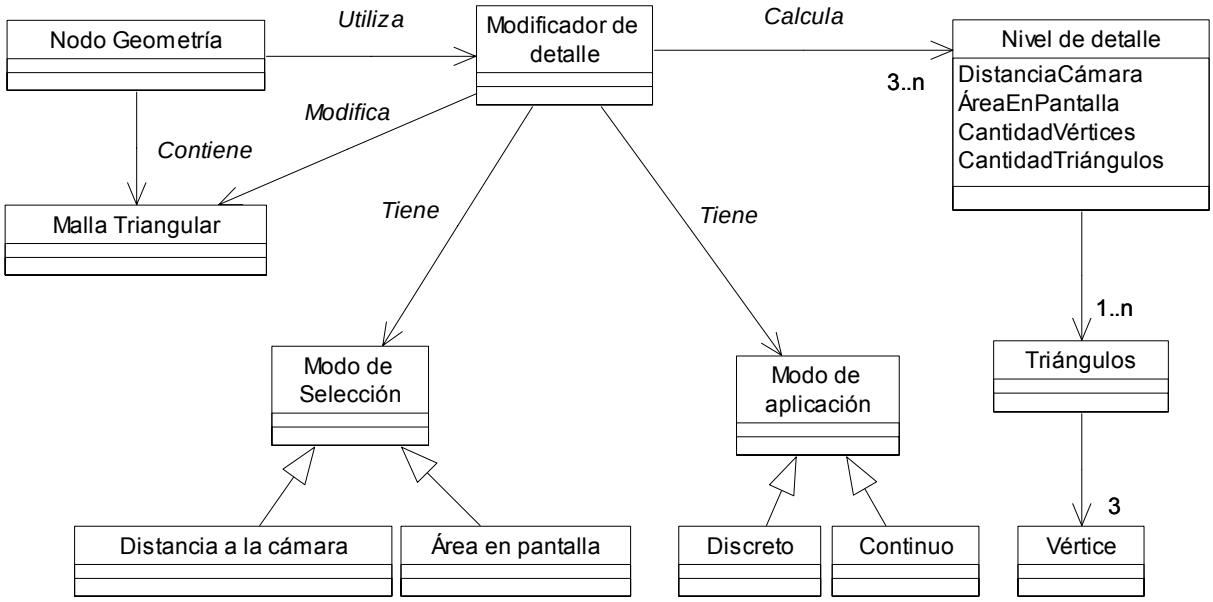



**Fig. 20 Modelo del dominio**

### 2.3.1 Glosario de términos del modelo del dominio

Este glosario de términos del dominio abarca los principales conceptos que conforman el modelo de dominio favoreciendo el conocimiento y posterior comprensión de los términos manejados.

Los conceptos **Nodo Geometría** y **Malla Triangular** no pertenecen a la modelación realizada como parte de esta investigación, sino que están modelados con anterioridad en la herramienta y que son utilizadas o modificadas por el resto de las clases involucradas en este modelo.

**Modificador de Detalle:** contiene los espectros o modelos de detalles, así como otros datos, determina y aplica el detalle a las geometrías.

**Nodo Geometría:** entidad que contiene a las geometrías y les da ubicación espacial en la escena.

**Malla Triangular:** estructura que almacena los datos de los vértices y sus conectividades.

**Nivel de Detalle:** nivel de optimización con que se visualizan las geometrías.

**Modo de Selección:** método utilizado por la aplicación para determinar el detalle óptimo.

**Modo de Aplicación:** método utilizado por la aplicación para aplicar el detalle óptimo.

## 2.4 Captura de requisitos

Los siguientes requisitos definirán el comportamiento del módulo a implementar y facilitarán el entendimiento con los usuarios y clientes.

### 2.4.1 Requisitos funcionales

Los requisitos funcionales enumerados a continuación representan el conjunto de funcionalidades y operaciones soportadas por el módulo:

1- Especificar la cantidad de polígonos a partir de la cual a los objetos se les aplicará niveles de detalles.
2- Especificar la cantidad mínima de polígonos que tendrán los objetos a deformar.
3- Definir el porciento de histéresis.
   3.1- Verificar que el valor de histéresis es menor que 100%.
   3.2- Asignar el valor de histéresis 10% por defecto.
4- Configurar zonas de detalles para LOD discreto (cercana, media y lejana).
   4.1- Verificar que la zona cercana sea menor que la mediana, y la mediana menor que la lejana.
5- Configurar límites de zonas de detalles para LOD continuo (cerca y lejos).
   5.1- Verificar que la zona cercana sea menor que la lejana.
6- Crear modificador de niveles de detalles.
   6.1- Verificar que el objeto no tenga ningún modificador asociado.
   6.2- Especificar la malla como parámetro.
   6.3- Especificar si se ignora o no la cantidad de polígonos cuando se le aplica nivel de detalles a un objeto.
   6.4- Especificar modo de aplicación (discreto o continuo).
   6.5- Inicializar la cantidad de polígonos para aplicar LOD.
   6.6- Inicializar la cantidad mínima de polígonos que tendrán los objetos a deformar.
   6.7- Inicializar el porciento de histéresis.
   6.8- Inicializar las zonas de detalles para LOD discreto (cercana, mediana, lejana).
   6.9- Inicializar los límites de zonas de detalles para LOD continuo (cerca, lejos).
   6.10- Inicializar el valor del nivel actual en cero.
7- Asociar modificador de LOD a un objeto.
8- Generar modelos de detalles.

8.1- Especificar la malla del objeto.

8.2- Crear listas auxiliares de triángulos, aristas y vértices.

8.3- Insertar cada dato de la malla original en las listas auxiliares correspondientes.

8.4- Crear lista principal de aristas.

8.5- Insertar cada arista de la lista auxiliar en la lista principal.

8.6- Calcular y almacenar métrica de fidelidad para cada arista en la lista principal de aristas.

8.7- Ordenar lista principal de aristas por métricas de fidelidad.

8.8- Colapsar la malla.

8.8.1- Verificar para cada arista de la lista principal de aristas si eliminar una arista no provoca pliegues.

8.8.2- Eliminar los triángulos compartidos por la arista a colapsar.

8.8.3- Crear nuevo espectro con la información de colapsamiento.

8.8.4- Insertar nuevo espectro en la lista de espectros.

8.8.5- Especificar la métrica de fidelidad como infinito si eliminar la arista provoca pliegues.

8.8.6- Reinsertar ordenado en la lista principal de aristas.

8.9- Buscar en la lista de espectros, la posición POS del primer espectro con más polígonos que el límite de colapsamiento.

8.10- Truncar la lista de espectros eliminando desde la posición POS hasta el final de la lista.

8.11- Destruir listas auxiliares de triángulos, aristas y vértices, y lista principal de aristas.

8.12- Generar el espectro de niveles de detalles para LOD continuo.

8.12.1- Calcular separación entre los modelos del espectro.

8.12.2- Guardar la distancia a la cámara para cada modelo.

8.13- Generar los 3 modelos de detalles para LOD discreto.

8.13.1- Crear las 3 nuevas mallas.

8.13.2- Crear el modelo cercano, especificando la malla y la distancia cercana.

8.13.3- Deformar la malla para nivel medio.

8.13.4- Crear el modelo medio, especificando la malla para nivel medio y la distancia media.

8.13.5- Deformar la malla para nivel lejano.

8.13.6- Crear el modelo lejano, especificando la malla para nivel lejano y la distancia lejana.

8.13.7- Guardar los 3 modelos.

9- Actualizar la malla de un objeto.

9.1- Verificar que la malla tiene asociado un modificador.

9.2- Actualizar el nivel actual.

10- Determinar nivel de detalle para un objeto según su distancia a la cámara.

10.1- Calcular distancia del modelo a la cámara.

10.2- Buscar el modelo adecuado comparando la distancia del modelo a la cámara y la distancia del objeto a modificar a la cámara.

10.3- Determinar según la histéresis si se aplica el nuevo nivel encontrado o se mantiene el actual.

10.4- Retornar el índice del nivel resultante.

11- Aplicar nivel de detalle a un objeto.

11.1- Verificar si el nivel nuevo y el existente son diferentes.

11.2- Sustituir la malla original completa con la malla N.

11.3- Sustituir la cantidad de vértices de la malla original con los del nivel N almacenado en el espectro.

11.4- Sustituir la cantidad de triángulos de la malla original con los del nivel N almacenado en el espectro.

11.5- Sustituir la lista de conectividades de los triángulos de la malla original con la lista del nivel N almacenado en el espectro.

## 2.4.2 Requisitos no funcionales

Los siguientes requisitos imponen condiciones al módulo a desarrollar, que no responden directamente a acciones pero regulan su comportamiento. Se relacionan aspectos como la usabilidad, rendimiento, disponibilidad, fiabilidad, seguridad, compatibilidad con hardware o software, etc.

**Usabilidad**: Los usuarios finales tendrán conocimientos básicos de gráfica computacional y programación, manejarán términos relacionados con el entorno gráfico. Se mantendrán los estándares de codificación usados en la herramienta base.

Debe ser flexible logrando que los usuarios finales adapten cada técnica de LOD implementada a su situación particular.

**Rendimiento**: La selección del nivel de detalle adecuado se ejecutará en tiempo real exigiendo un alto grado de velocidad en el cálculo de la cantidad de polígonos aplicables al objeto, alta disponibilidad a los usuarios de la herramienta, así como un rápido tiempo de respuesta y recuperación.

**Soporte:** En una primera versión la plataforma en la que correrá será Windows, pero debe tener características que permitan una fácil y rápida migración a la plataforma Linux.

**Hardware:** Debe existir compatibilidad con las tarjetas gráficas especificadas por los usuarios, son las pertenecientes a la familia NVIDIA (Quadro FX 500/FX 600, Geforce 3, Geforce 4, FX 5200).

**Diseño e Implementación**: La filosofía de programación del módulo será Orientada a Objetos, acorde a las características de la herramienta de la que formará parte. Se utilizará el lenguaje de programación C++.

**Legales**: Se regirá por las normas ISO 690.

**Confiabilidad**: Se realizarán todas las validaciones necesarias para impedir un "crash" del sistema, es decir todo lo relacionado con la seguridad del mismo y su funcionamiento. También se protegerá del mal manejo por parte de los usuarios.

**Interfaz**: Tendrá una interfaz de programación con código legible, adaptado al código de la herramienta base, que permita acceso a las funcionalidades del módulo, pero manteniendo alejado al usuario del funcionamiento interno. El módulo en si mismo no tendrá una interfaz visual.

**Portabilidad:** El objetivo es que la implementación del módulo pueda ser aplicada a múltiples plataformas.

**Software:** Se utilizará el IDE Microsoft Visual Studio .NET 2003 sobre el sistema operativo Windows 2003.

# 2.5 Modelo de casos de usos del sistema

## 2.5.1 Actores del sistema

Un actor es un usuario del sistema, que usa un caso de uso para desempeñar alguna porción de trabajo que es de valor para el negocio. El conjunto de casos de uso al que un actor tiene acceso define su rol global en el sistema y el alcance de su acción. En el módulo interactúa solo un actor, se trata del programador.

**Tabla 1 Actor del Sistema**

| Actor | Justificación |
|---|---|
| Programador | Va a ser quien se favorecerá con las funcionalidades presentes en el módulo, es decir es quién podrá de manera general: configurar la aplicación, asociar modificadores de LOD a los objetos, actualizar la malla y generar detalles. |

## 2.5.2 Descripción de casos de usos

Cada caso de uso tiene una descripción que describe la funcionalidad que se construirá en el módulo propuesto. Un caso de uso puede "incluir" la funcionalidad de otro caso de uso con su propio comportamiento, en el modelo de casos de uso creado se representan relaciones de este tipo.

Descripciones de casos de uso propuestos:

**Tabla 2 Descripción del CU Configurar aplicación**

| | |
|---|---|
| CU-1 | Configurar aplicación |
| Actor | Programador |
| Descripción | Permite especificar cuando se inicia la aplicación, los parámetros que regirán el proceso de creación, selección y aplicación de los niveles de detalles. |
| Referencia | RF 1, 2, 3, 3.1, 3.2, 4, 4.1, 5, 5.1. |

**Tabla 3 Descripción del CU Asociar modificador a un objeto**

| | |
|---|---|
| CU-2 | Asociar modificador a un objeto |
| Actor | Programador |
| Descripción | Permite crear y asociarle un modificador de niveles de detalles a un objeto. |
| Referencia | RF 6, 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 6.10, 7 y CU incluido “Generar detalles”. |

**Tabla 4 Descripción del CU Generar detalles**

| | |
|---|---|
| CU-3 | Generar Detalles |
| Actor | CU Asociar modificador a un objeto |
| Descripción | Dada la malla de un objeto, genera el espectro de detalles para la misma. |
| Referencia | RF 8, 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8, 8.8.1, 8.8.2, 8.8.3, 8.8.4, 8.8.5, 8.8.6, 8.9, 8.10, 8.11, 8.12, 8.12.1, 8.12.2, 8.13, 8.13.1, 8.13.2, 8.13.3, 8.13.4, 8.13.5, 8.13.6, 8.13.7 |

**Tabla 5 Descripción del CU Seleccionar nivel de detalle**

| | |
|---|---|
| CU-4 | Seleccionar Nivel de detalle |
| Actor | Programador |
| Descripción | Determina el nivel de detalle para un objeto según su distancia a la cámara. |
| Referencia | RF 10, 10.1, 10.2, 10.3, 10.4 |

**Tabla 6 Descripción del CU Aplicar nivel de detalle**

| | |
|---|---|
| CU-5 | Aplicar Nivel de detalle |
| Actor | Programador |
| Descripción | Le aplica el nivel de detalle a un objeto, dado el nivel N a aplicar. |
| Referencia | RF 11, 11.1, 11.2, 11.3, 11.4, 11.5 |

**Tabla 7 Descripción del CU Actualizar malla**

| | |
|---|---|
| CU-6 | Actualizar malla |
| Actor | Programador |
| Descripción | La malla se actualiza a partir de la aplicación del nivel de detalle correspondiente. |
| Referencia | RF 9, 9.1, 9.2 y CU incluido “Seleccionar Nivel de detalle” y “Aplicar Nivel de detalle”. |

### 2.5.3 Modelo de casos de usos

El Lenguaje de Modelado Unificado (UML) define una notación gráfica para representar casos de uso llamada modelo de casos de uso. El modelo de casos de uso describe la funcionalidad propuesta del nuevo sistema, el módulo en este caso, apoyando la captura de los requisitos funcionales. El siguiente modelo funciona como una notación gráfica que da una vista general simple del conjunto de casos de uso que representan las funcionalidades o procesos principales que se operarán en el módulo, las dependencias entre ellos, así como el actor que interactúa con el sistema.

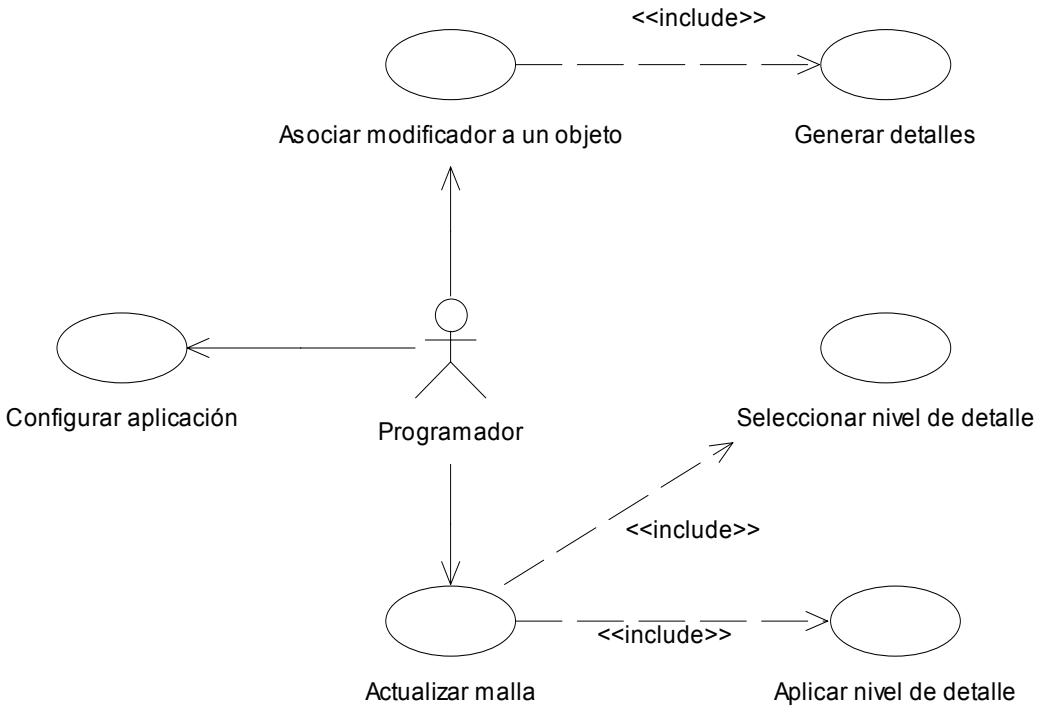



**Fig. 21 Modelo de Casos de Usos del Sistema**

### 2.5.4 Descripción expandida de casos de uso

Un caso de uso expandido muestra más detalles que un caso de uso de alto nivel. Los casos de uso expandidos son útiles para alcanzar un conocimiento más profundo de los procesos y los requerimientos.

**Tabla 8 Expansión del CU Configurar aplicación**

| Caso de uso | |
|---|---|
| CU-1 | Configurar aplicación |
| Propósito | Especificar los parámetros que regirán el proceso de creación, selección y aplicación de los niveles de detalles. |
| Actores | Programador (Inicia) |
| Resumen: El caso de uso se inicia cuando el programador solicita configurar algún parámetro del módulo de niveles de detalles. Se le otorga el valor al parámetro especificado. El caso de uso finaliza cuando la aplicación esté configurada. | |
| Referencias | RF 1, 2, 3, 3.1, 3.2, 4, 4.1, 5, 5.1 |
| Precondiciones | |
| Postcondiciones | Debe quedar configurada la aplicación con todos los parámetros. |
| Curso Normal de los Eventos | |
| Acción del actor | Respuesta del sistema |
| 1- El programador solicita configurar un parámetro.<br>a) Si es la cantidad de polígonos para aplicar LOD ir a la “sección 1”<br>b) Si es la cantidad mínima de polígonos que tendrán los objetos a deformar ir a la “sección 2”<br>c) Si es el porciento de histéresis ir a la “sección 3”.<br>d) Si son las zonas de detalles para LOD discreto ir a la “sección 4”.<br>e) Si son los límites de zonas de detalles para LOD continuo ir a la “sección 5”. | |

| Sección 1: Cantidad de polígonos para aplicar LOD | |
|---|---|
| Acción del actor | Respuesta del sistema |
| 1- Se pasa el valor de la cantidad de polígonos para aplicar LOD. | |
| | 1.1- Se asigna el valor de la cantidad de polígonos para aplicar LOD. |
| Sección 2: Cantidad mínima de polígonos | |
| Acción del actor | Respuesta del sistema |
| 1- Se pasa el valor mínimo de polígonos. | |
| | 1.1- Se asigna el valor mínimo de polígonos que tendrán los objetos a deformar. |
| Sección 3: Porciento de histéresis | |
| Acción del actor | Respuesta del sistema |
| 1- Se pasa el valor del porciento de histéresis | |
| | 1.1- Si el valor es menor que 100%, se asigna el valor de histéresis especificado. |
| Curso Alterno de los Eventos | |
| | 1.1- Se asigna el valor 10% por defecto. |
| Sección 4: Zonas de detalles Discreto | |
| Acción del actor | Respuesta del sistema |
| 1- Se pasan las zonas de detalle (cercana, mediana, lejana). | |
| | 1.1- Si los valores cumplen que: la zona cercana sea menor que la mediana, y la mediana menor que la lejana, se asignan las zonas de detalle para LOD discreto (cercana, mediana, lejana). |
| Curso Alterno de los Eventos | |
| | 1.1- No se asignan los valores. |
| Sección 5: Límites de zonas de detalles Continuo | |

| Acción del actor | Respuesta del sistema |
|---|---|
| 1- Se pasan los límites de las zonas de detalle (cerca y lejos). | |
| | 1.1- Si los valores cumplen que: la zona cercana sea menor que la lejana, se asignan las zonas de detalle para LOD continuo (cerca y lejos). |
| Curso Alterno de los Eventos | |
| | 1.1- No se asignan los valores. |

**Tabla 9 Expansión del CU Asociar modificador a un objeto**

| Caso de uso | |
|---|---|
| CU-2 | Asociar modificador a un objeto |
| Propósito | Asociarle modificadores de niveles de detalles a los objetos. |
| Actores | Programador (Inicia) |
| Resumen: El caso de uso inicia cuando el programador solicita la creación y asociación de un modificador a un objeto. El caso de uso finaliza cuando el objeto tiene asociado un modificador. | |
| Referencias | RF 6, 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 6.10, 7 y CU incluido “Generar detalles”. |
| Precondiciones | No debe tener ningún modificador asociado. |
| Postcondiciones | Se crea y asocia el modificador, el objeto queda listo para que se le modifique la malla por niveles de detalles. |
| Curso Normal de los Eventos | |
| Acción del actor | Respuesta del sistema |
| 1- Llama al método de asociar un modificador y especifica el objeto a modificar, y si se ignora o no la cantidad de polígonos para aplicar LOD. | |
| | 1.1- Si se especificó que se ignore la cantidad de polígonos para aplicar LOD, o no se |

| | |
|---|---|
| | especificó y la cantidad de polígonos del objeto pasado por parámetros es mayor que la cantidad de polígonos para aplicar LOD, se crea el modificador. |
| | 1.2- Se inicializa la cantidad de polígonos para aplicar LOD. |
| | 1.3- Se inicializa la cantidad mínima de polígonos que tendrán los objetos a deformar. |
| | 1.4- Se inicializa el modo de aplicación. |
| | 1.5- Se inicializa el porciento de histéresis. |
| | 1.6- Se inicializan las zonas de detalles para LOD discreto (cercana, mediana, lejana). |
| | 1.7- Se inicializan límites de zonas de detalles para LOD continuo (cerca, lejos). |
| | 1.8- Inicializa el valor del nivel actual en cero. |
| | 1.9- Invoca al CU “Generar detalles” pasando como parámetro la malla del objeto. |
| | 1.10- Asocia el modificador obtenido al objeto. |
| Curso Alterno de los Eventos | |
| | 1.1- No se crea ni se asocia el modificador. |

**Tabla 10 Expansión del CU Generar detalles**

| Caso de uso | |
|---|---|
| CU-3 | Generar Detalles (Incluido de CU “Asociar modificador a un objeto”) |
| Propósito | Genera el espectro de niveles de detalles de la malla de un objeto. |
| Actores | CU Asociar Modificador a un objeto (Inicia) |
| Resumen: El caso de uso se inicia cuando el programador o el caso de uso “Asociar modificador a un objeto” solicitan la generación del espectro de detalles especificando la malla de un objeto. El CU termina cuando se crea el espectro y se retorna. | |
| Referencias | RF 8, 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8, 8.8.1, 8.8.2, 8.8.3, 8.8.4, 8.8.5, 8.8.6, 8.9, 89.10, 8.11, 8.12, 8.12.1, 8.12.2, 8.13, 8.13.1, 8.13.2, 8.13.3, 8.13.4, 8.13.5, 8.13.6, 8.13.7 |

| | |
|---|---|
| Precondiciones | |
| Postcondiciones | Deben quedar creados el espectro de detalles continuos o los 3 modelos discretos. |
| Curso Normal de los Eventos | |
| Acción del actor | Respuesta del sistema |
| 1- Solicita generar el espectro especificando la malla del objeto. | |
| | 1.1- Crear listas auxiliares de triángulos, aristas y vértices. |
| | 1.2- Insertar cada dato de la malla original (triángulos, aristas y vértices) en las listas auxiliares correspondientes. |
| | 1.3- Crear lista principal de aristas. |
| | 1.4- Insertar cada arista de la lista auxiliar en la lista principal. |
| | 1.5- Calcular y almacenar métrica de fidelidad para cada arista en la lista principal de aristas. |
| | 1.6- Ordenar lista principal de aristas por métricas de fidelidad. |
| | 1.7- Colapsar la malla (colapsar cada arista de la malla). Para cada arista de la lista principal de aristas: |
| | 1.7.1- Si eliminar la arista no provoca pliegues, eliminar los triángulos compartidos por la arista a colapsar (eliminar los datos triángulo, aristas y vértices correspondientes de las estructuras de triángulos, aristas y vértices auxiliares). |
| | 1.7.2- Crear nuevo espectro con la información de colapsamiento (vértice a mantener, vértice a eliminar, cantidad de vértices y de triángulos que quedan después |

| | |
|---|---|
| | del colapsamiento, y lista de índices que especifican las conectividades de los triángulos de la nueva malla). |
| | 1.7.3- Insertar nuevo espectro en la lista de espectros. Si quedan aristas regresar a 1.7.1. |
| | 1.8- Buscar en la lista de espectros, la posición POS del primer espectro con más polígonos que el límite de colapsamiento. |
| | 1.9- Truncar la lista de espectros eliminando desde la posición POS hasta el final de la lista. |
| | 1.10- Destruir listas auxiliares de triángulos, aristas y vértices, y lista principal de aristas. |
| | 1.11- Si el modo de aplicación es continuo, calcular separación entre los modelos del espectro. |
| | 1.12- Para cada modelo, guardar la distancia a la cámara (producto de la separación entre modelos por la posición del modelo en la lista). |
| Curso Alterno de los Eventos “Provoca pliegues”. | |
| Acción del actor | Respuesta del sistema |
| | 1.7.1- Si eliminar la arista provoca pliegues, se le pone métrica de fidelidad infinito. |
| | 1.7.2- Se reinserta ordenado en la lista principal de aristas y se regresa a 1.7.1 en el curso normal de los eventos. |
| Curso Alterno de los Eventos “Modo discreto”. | |
| Acción del actor | Respuesta del sistema |
| | 1.11- Si el modo de aplicación es discreto, se crean las 3 nuevas mallas. |
| | 1.12- Se crea el modelo cercano, especificando la malla y la distancia cercana. |

| | |
|---|---|
| | 1.13- Se deforma la malla para nivel medio. |
| | 1.14- Se crea el modelo medio, especificando la malla para nivel medio y la distancia media. |
| | 1.15- Se deforma la malla para nivel lejano. |
| | 1.16- Se crea el modelo lejano, especificando la malla para nivel lejano y la distancia lejana. |
| | 1.17- Se guardan los 3 modelos. |

**Tabla 11 Expansión del CU Actualizar malla**

| Caso de uso | |
|---|---|
| CU-4 | Actualizar Malla |
| Propósito | Actualizar la malla según el nivel de detalle correspondiente. |
| Actores | Programador (Inicia) |
| Resumen: El caso de uso se inicia con la verificación de que a la malla se le está aplicando LOD, para en caso afirmativo seleccionar y aplicar el nivel adecuado. Este caso de uso finaliza cuando la malla no tiene asociado ningún modificador o con la aplicación final del nivel N adecuado. | |
| Referencias | RF 9, 9.1, 9.2 y CU incluido “Seleccionar Nivel de detalle” y “Aplicar Nivel de detalle”. |
| Precondiciones | El objeto que contiene la malla debe tener asociado un modificador. |
| Postcondiciones | La malla debe quedar actualizada con el nivel de detalle seleccionado. |
| Curso Normal de los Eventos | |
| Acción del actor | Respuesta del sistema |
| 1- Solicita actualizar estado de la malla. | |
| | 1.1- Si la malla tiene asociado un modificador, invoca al CU “Seleccionar nivel de detalle” que se encarga de seleccionar el nivel N adecuado para el objeto. |
| | 1.2- Se invoca al CU “Aplicar nivel de detalle” |

| | |
|---|---|
| | que se encarga de aplicar el nivel de detalle N al objeto. |
| | 1.3- Se actualiza el nivel actual. |
| Curso Alterno de los Eventos | |
| Acción del actor | Respuesta del sistema |
| | 1.2- Si no tiene asociado un modificador finaliza el caso de uso. |

**Tabla 12 Expansión del CU Seleccionar nivel de detalle**

| Caso de uso | |
|---|---|
| CU-5 | Seleccionar nivel de detalle (Incluido de CU "Actualizar Malla"). |
| Propósito | Determinar el nivel de detalle para un objeto según su distancia a la cámara. |
| Actores | CU Actualizar Malla (Inicia) |
| Resumen: El caso de uso se inicia cuando el CU-4 "Actualizar malla" solicita seleccionar el nivel de detalle. A partir del conocimiento de la distancia del objeto a la cámara, para luego aplicar el nivel adecuado a dicha distancia y considerando la histéresis para niveles de detalles discretos. El caso de uso finaliza con el retorno del índice del nivel a aplicar. | |
| Referencias | RF 10, 10.1, 10.2, 10.3, 10.4 |
| Precondiciones | El objeto que contiene la malla debe tener asociado un modificador. |
| Postcondiciones | El nivel resultante de la operación debe ser retornado. |
| Curso Normal de los Eventos | |
| Acción del actor | Respuesta del sistema |
| 1- Se solicita calcular el nivel a aplicar pasando la posición de la cámara y del objeto a transformar. | |
| | 1.1- Calcular distancia del modelo a la cámara. |
| | 1.2- Buscar el modelo adecuado comparando la distancia del modelo a la cámara y la distancia del objeto a modificar a la cámara. |

| | |
|---|---|
| | 1.3- Si el modo de aplicación es discreto, determinar según la histéresis si se aplica el nuevo nivel encontrado o se mantiene el actual. |
| | 1.4- Retorna el índice del nivel resultante. |
| Curso Alterno de los Eventos | |
| Acción del actor | Respuesta del sistema |
| | 1.3- Si el modo de aplicación es continuo no se aplica histéresis. Ir a 1.4. |

**Tabla 13 Expansión del CU Aplicar nivel de detalle**

| Caso de uso | |
|---|---|
| CU-6 | Aplicar Nivel de Detalle (Incluido de CU “Actualizar Malla”). |
| Propósito | Actualizar la malla de un objeto dado el nivel N a aplicar. |
| Actores | CU Actualizar Malla (Inicia) |
| Resumen: El caso de uso inicia cuando el CU-4 “Actualizar malla” solicita aplicar el nivel de detalle. Dado el nivel N que se aplicará al objeto, los pasos siguientes dependen del modo de aplicación (discreto o continuo). El caso de uso finaliza si el nivel a aplicar es el mismo que el existente (en ese caso no se modifica la malla), o con la modificación de la malla según el nuevo nivel. | |
| Referencias | RF 11, 11.1, 11.2, 11.3, 11.4, 11.5 |
| Precondiciones | El objeto que contiene la malla debe tener asociado un modificador. |
| Postcondiciones | La malla debe quedar actualizada con el nivel de detalle indicado. |
| Curso Normal de los Eventos | |
| Acción del actor | Respuesta del sistema |
| 1- Se pasa el índice N del nivel a aplicar. | |
| | 1.1- Si el nivel nuevo y el existente son diferentes, y el modo de aplicación es discreto, se sustituye la malla original completa con la malla N. |

| Curso Alterno de los Eventos | |
|---|---|
| Acción del actor | Respuesta del sistema |
| | 1.1- Si el nivel nuevo y el existente son diferentes, y el modo de aplicación es continuo, se sustituye la cantidad de vértices de la malla original con los del nivel N almacenado en el espectro. |
| | 1.2- Se sustituye la cantidad de triángulos de la malla original con los del nivel N almacenado en el espectro. |
| | 1.3- Se sustituye la lista de conectividades de los triángulos de la malla original con la lista del nivel N almacenado en el espectro. |
| Curso Alterno de los Eventos | |
| Acción del actor | Respuesta del sistema |
| | 1.1- Si el nivel nuevo y el existente son iguales no se cambia el detalle. |

## Conclusiones

Al finalizar este capítulo ya están definidas las bases técnicas que gobernarán al módulo. Quedó especificado el resultado que esperan los futuros usuarios del sistema, así como los requisitos funcionales de éste y las descripciones de los casos de uso que harán posible la obtención del resultado.

# 3

# Capítulo 3 Diseño del sistema

## Introducción

Este capítulo comienza mostrando el diseño del sistema propuesto. Se representan las clases del diseño, las responsabilidades de cada una y las relaciones entre ellas, las clases están agrupadas por paquetes mostrando el resultado ya pulido de las etapas antecesoras.

Se representan los artefactos involucrados en el diseño del módulo, es decir los diagramas clase y secuencia de la realización de los casos de uso.

## 3.1 Diagrama de clases del diseño

A continuación algunas aclaraciones para una mayor comprensión de dicho diagrama:

1- La nomenclatura utilizada para los diagramas de clases, se explica en el epígrafe “Estándares de codificación” del siguiente capítulo.
2- En las clases representadas en los diagramas los métodos de acceso a miembros de clases (“gets” y “sets”), constructores por defecto, constructores de copia y destructores, son omitidos. En algunos casos se incluyen algunos constructores, pero no son los “por defecto”.
3- El contenido de los paquetes de clases del diseño se muestra en el los diagramas de clases del diseño correspondientes.

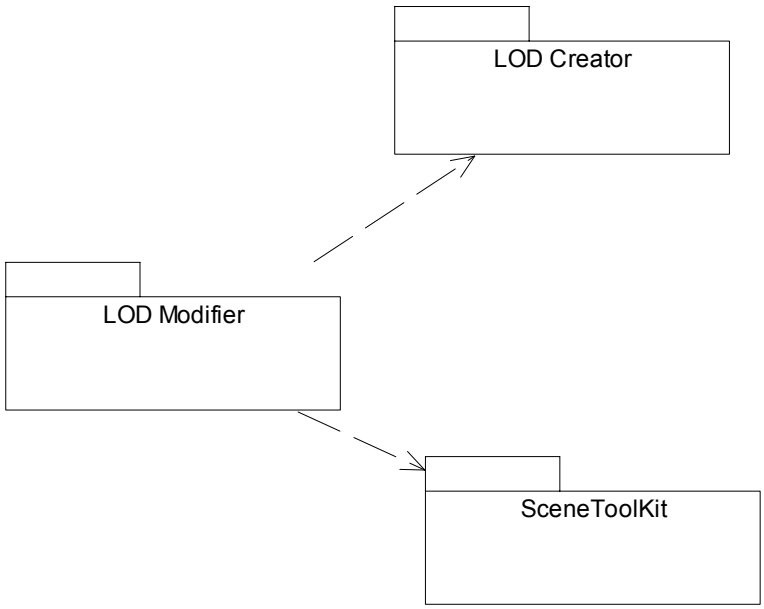



**Fig. 22 Diagrama de Paquetes de Clases del Diseño**

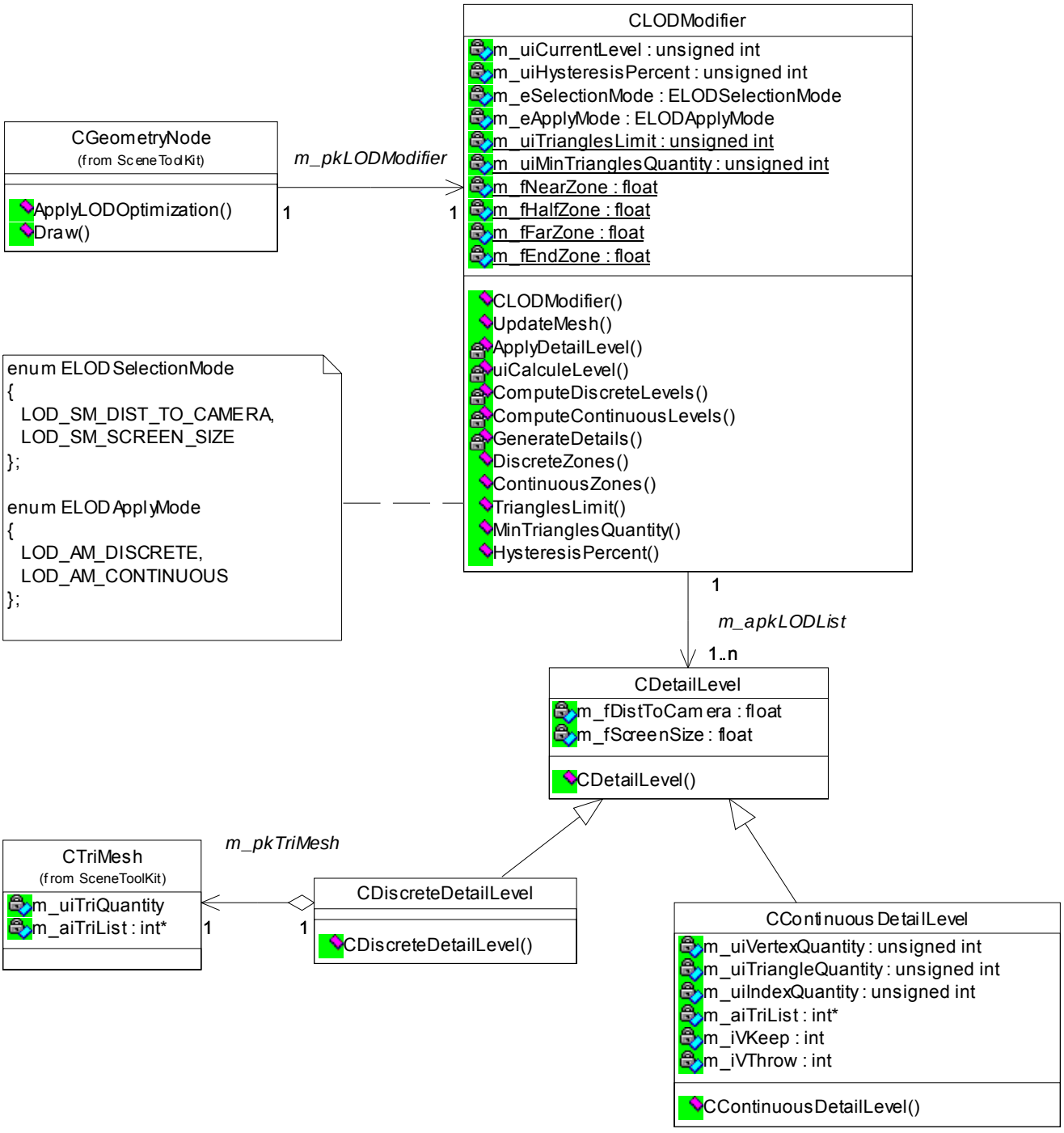



**Fig. 23 Diagrama de Clases del Diseño "LODModifier"**

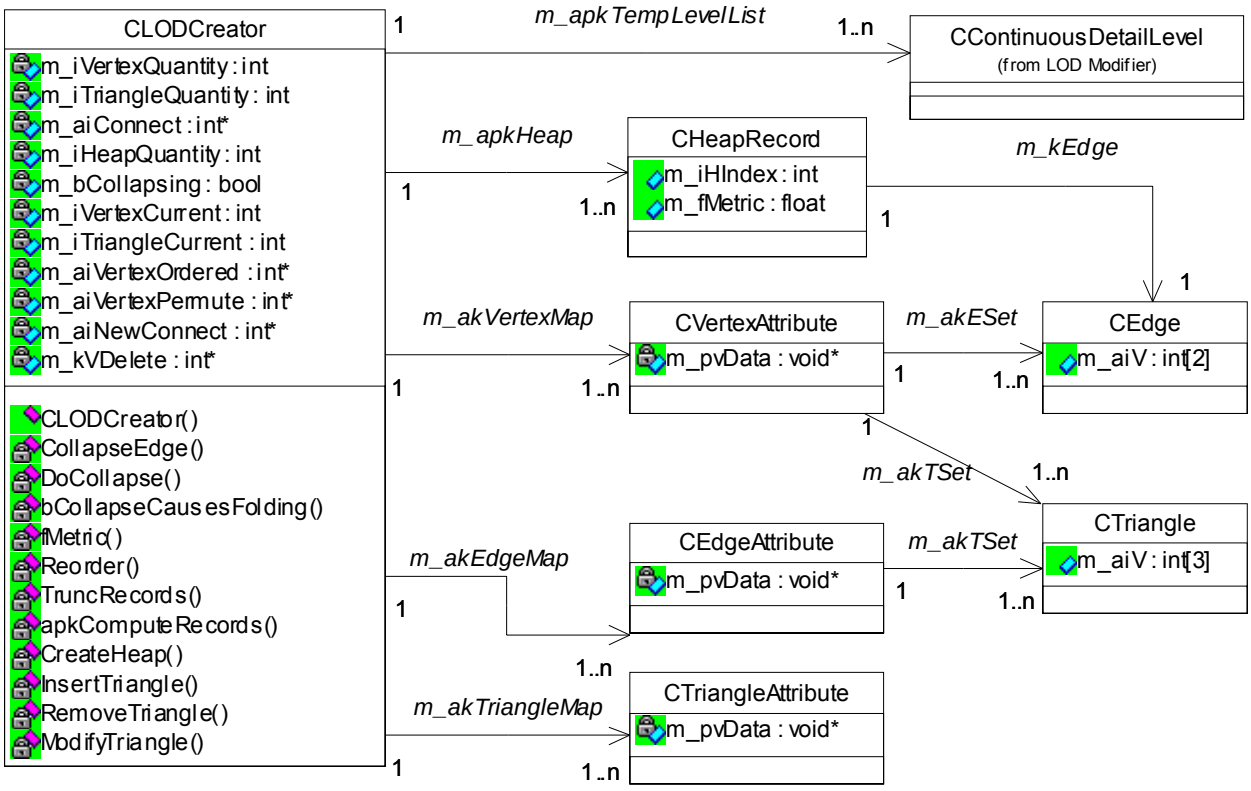



**Fig. 24 Diagrama de Clases del Diseño "LODCreator"**

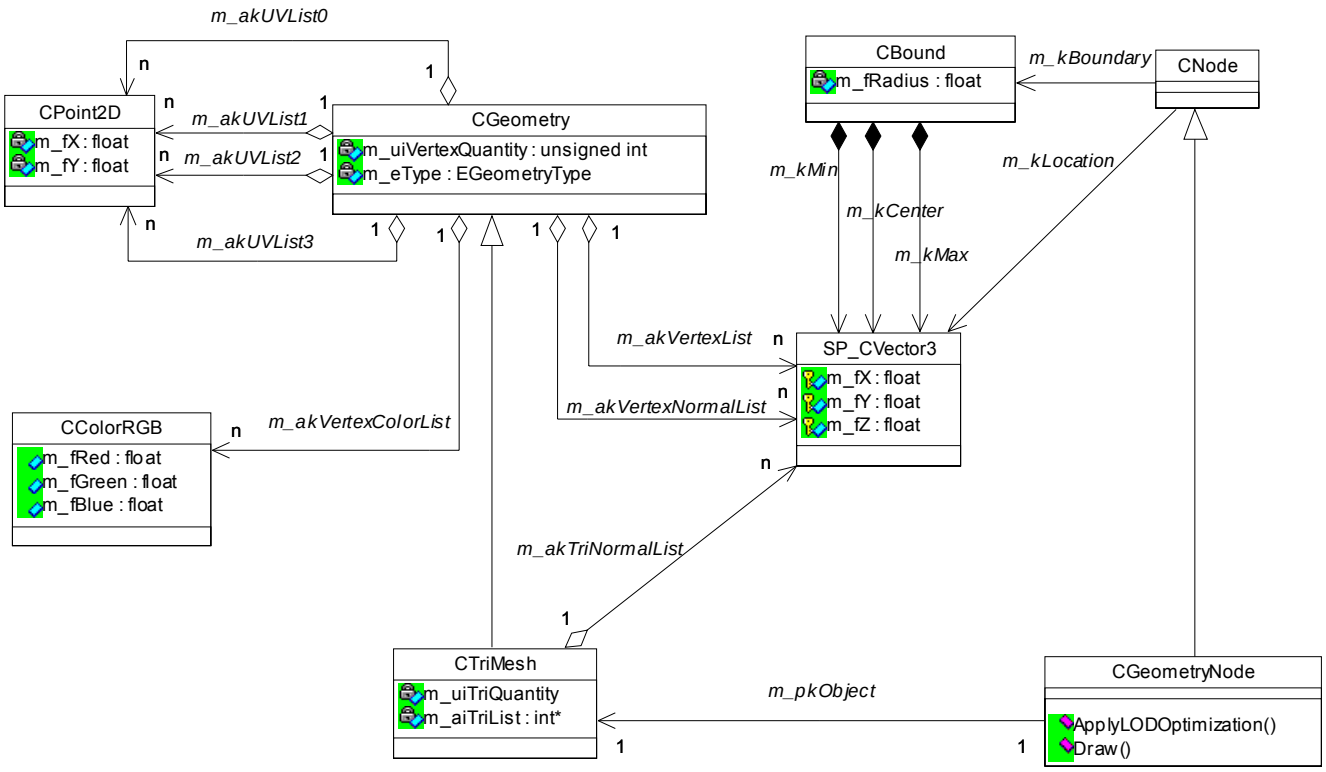



**Fig. 25 Diagrama de Clases del Diseño "SceneToolKit"**

## 3.1.1 Descripción de las clases del diseño

Las descripciones que aparecen a continuación son las de las clases con mayor importancia dentro del diseño de clases realizado. Las clases con un nivel de relevancia bajo no serán descritas. No serán descritas las clases del diseño pertenecientes a la herramienta "Scene Toolkit".

**Tabla 14 Descripción de la clase "CDetailLevel"**

| **Nombre:** CDetailLevel | |
|---|---|
| **Tipo de clase:** Entidad | |
| **Atributo** | **Tipo** |
| m_fDistToCamera | float |
| m_fScreenSize | float |
| **Para cada responsabilidad**: | |
| Nombre: | **CDetailLevel** (float arg_fDistToCamera , float arg_fScreenSize ) |
| Descripción: | Constructor de la clase. |

**Tabla 15 Descripción de la clase "CContinuousDetailLevel"**

| **Nombre:** CContinuousDetailLevel | |
|---|---|
| **Tipo de clase:** Entidad | |
| **Atributo** | **Tipo** |
| m_uiVertexQuantity | unsigned int |
| m_uiTriangleQuantity | unsigned int |
| m_uiIndexQuantity | unsigned int |
| m_aiIndexList | int* |
| m_iVKeep | int |
| m_iVThrow | int |
| **Para cada responsabilidad**: | |
| Nombre: | **CContinuousDetailLevel** (int arg_iVKeep, int arg_iVThrow, unsigned int arg_uiVertexQuantity , unsigned int arg_uiTriQuantity) |
| Descripción: | Constructor de la clase. |

**Tabla 16 Descripción de la clase "CDiscreteDetailLevel"**

| **Nombre:** CDiscreteDetailLevel | |
|---|---|
| **Tipo de clase:** Entidad | |
| **Atributo** | **Tipo** |
| m_pkTriMesh | CTriMesh* |
| **Para cada responsabilidad**: | |
| Nombre: | **CdiscreteDetailLevel** (CTriMesh* arg_pkTriMesh, float arg_fDistToCamera, float arg_fScreenSize ) |
| Descripción: | Constructor de la clase. |

**Tabla 17 Descripción de la clase "CLODCreator"**

| **Nombre:** CLODCreator | |
|---|---|
| **Tipo de clase :** Controladora | |
| **Atributo** | **Tipo** |
| m_iVertexQuantity | int |
| m_iTriangleQuantity | int |
| m_aiConnect | int* |
| m_iHeapQuantity | int |
| m_bCollapsing | bool |
| m_iVertexCurrent | int |
| m_iTriangleCurrent | int |
| m_aiVertexOrdered | int* |
| m_aiVertexPermute | int* |
| m_aiNewConnect | int* |
| m_kVDelete | int* |
| **Para cada responsabilidad**: | |
| Nombre: | **CLODCreator**(TContinuousLODList& arg_apkContinuousList, unsigned int arg_uiMinTrianglesQuantity) |
| Descripción: | El constructor de la clase. |
| Nombre: | **CollapseEdge**(arg_iVKeep, arg_iVThrow) |
| Descripción: | Colapsa la arista que contiene el keep y el throw que le pasan como parámetro y elimina los triángulos compartidos por el vértice a colapsar. |
| Nombre: | **DoCollapse**() |
| Descripción: | Encuentra qué arista es la que va a colapsar y el vértice throw que se va a eliminar. |
| Nombre: | **bCollapseCausesFolding**(arg_iVKeep, arg_iVThrow) |
| Descripción: | Verifica si al eliminar la arista que contiene estos vértices, se causa un pliegue en la malla. |
| Nombre: | **fMetric**(arg_kEdge) |
| Descripción: | Le calcula la métrica a la arista que le pasan como parámetro. |
| Nombre: | **Reorder**() |
| Descripción: | Reordena las listas paralelas de UV, normales, colores... |
| Nombre: | **TruncRecords**(arg_uiMinTrianglesQuantity) |
| Descripción: | Desecha del espectro continuo de niveles los modelos que tengan menos de la cantidad de triángulos que le pasan por parámetro. |
| Nombre: | **apkComputeRecords**() |
| Descripción: | Genera el espectro final del modelo continuo de detalles. |
| Nombre: | **CreateHeap**() |
| Descripción: | Se crea una lista de aristas con el doble de la capacidad de la original, ya que es posible que durante un colapsamiento de bordes el número de bordes "temporales" sea mayor que la cantidad original de la malla. |
| Nombre: | **InsertTriangle**(arg_iV0, arg_iV1, arg_iV2) |
| Descripción: | Inserta un triángulo en el TriangleMap |
| Nombre: | **RemoveTriangle**(arg_iV0 : int, arg_iV1 : int, arg_iV2 : int) |

| | |
|---|---|
| Descripción: | Elimina un triángulo de el TriangleMap |
| Nombre: | **ModifyTriangle**(arg_kTriangle, arg_iVKeep , arg_iVThrow) |
| Descripción: | El grupo de aristas potencialmente modificadas son todas las que son compartidas por los triángulos que contienen vértices 'keep'. Modifica estas métricas y actualiza el heap. |

**Tabla 18 Descripción de la clase "CLODModifier"**

| **Nombre:** CLODModifier | |
|---|---|
| **Tipo de clase:** Controladora | |
| **Atributo** | **Tipo** |
| m_uiCurrentLevel | unsigned int |
| m_uiHysteresisPercent | unsigned int |
| m_eSelectionMode | ELODSelectionMode |
| m_eApplyMode | ELODApplyMode |
| m_uiTrianglesLimit | unsigned int |
| m_uiMinTrianglesQuantity | unsigned int |
| m_fNearZone | float |
| m_fHalfZone | float |
| m_fFarZone | float |
| m_fEndZone | float |
| **Para cada responsabilidad**: | |
| Nombre: | **CLODModifier**(arg_pkTriMesh, arg_eSelecionMode ) |
| Descripción: | El constructor de la clase. |
| Nombre: | **UpdateMesh**(arg_pkTriMesh, arg_kViewerLocation, arg_kObjectLocation, arg_fObjectScreenSize) |
| Descripción: | Actualiza la malla con el nuevo nivel de detalle. |
| Nombre: | **ApplyDetailLevel**(arg_pkTriMesh, arg_uiLevel ) |
| Descripción: | Aplica a la malla el nivel de detalle entrado por parámetros. |
| Nombre: | **uiCalculeLevel**(arg_kViewerLocation, arg_kObjectLocation, arg_fObjectScreenSize) |
| Descripción: | De acuerdo a la distancia a la que se encuentra el objeto de la cámara busca el modelo más adecuado. |
| Nombre: | **ComputeDiscreteLevels**(arg_pkTriMesh) |
| Descripción: | Se crean tres mallas sin deformar, obtiene las tres nuevas mallas para nivel cercano, medio y lejano y guarda los modelos en la lista de niveles. |
| Nombre: | **ComputeContinuousLevels**(arg_pkTriMesh) |
| Descripción: | Ubica los modelos según la distancia a la cámara. |
| Nombre: | **GenerateDetails**(arg_pkTriMesh) |
| Descripción: | Manda a crear el espectro de continuo niveles de detalles. |
| Nombre: | **HysteresisPercent**(arg_uiValue) |
| | Aplica el porciento de histéresis especificado por parámetro, si no se especifica aplica por defecto el 10%. |

## 3.2 Diagramas de secuencia

A continuación siguen los diagramas de secuencias respondiendo a los diferentes escenarios de los casos de uso definidos en capítulos anteriores.

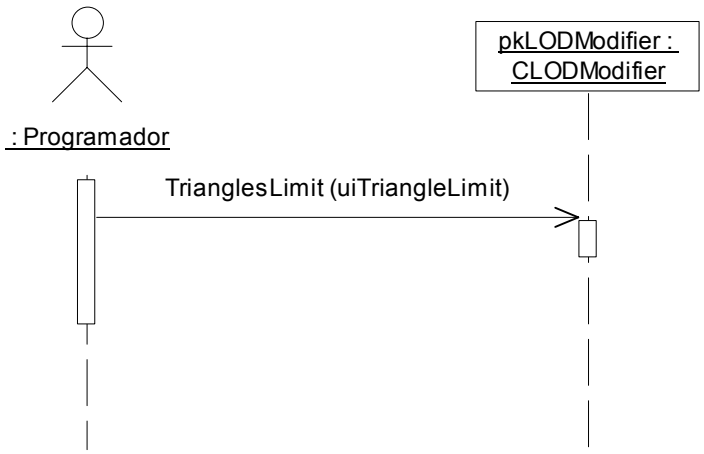



**Fig. 26 Diagrama de Secuencia "Configurar Aplicación" - escenario "Cantidad de polígonos para aplicar LOD"**

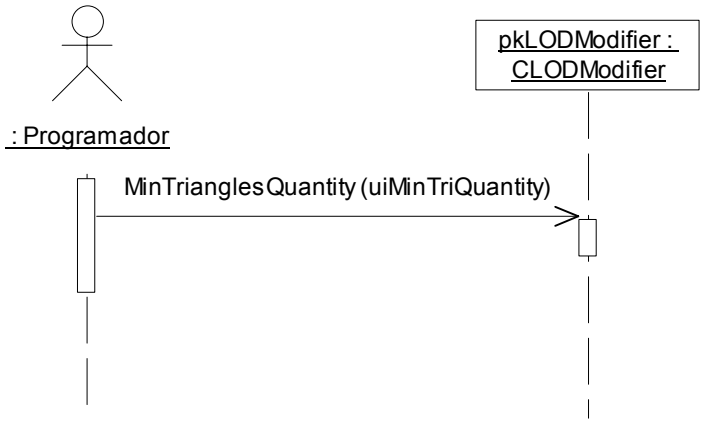



**Fig. 27 Diagrama de Secuencia "Configurar Aplicación" – escenario "Cantidad mínima de polígonos"**

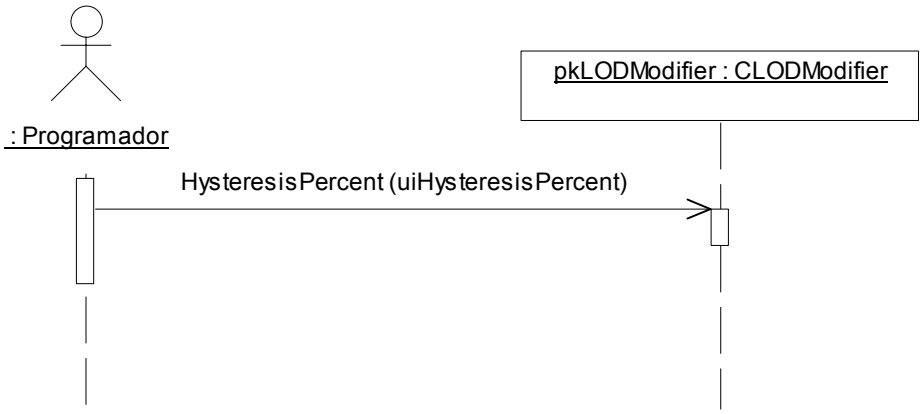



**Fig. 28 Diagrama de Secuencia "Configurar Aplicación" – escenario "Histéresis"**

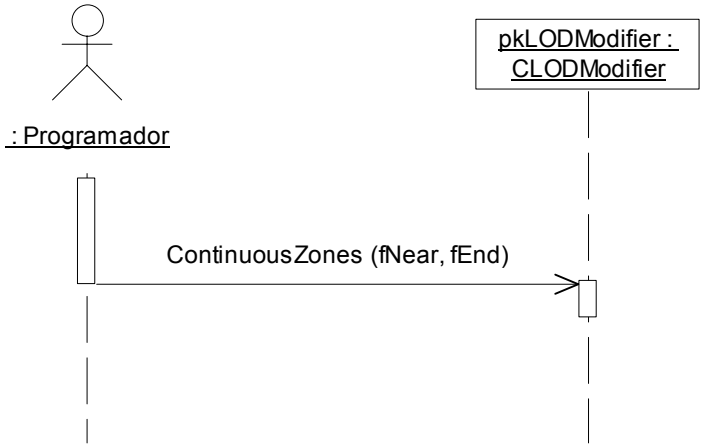



**Fig. 29 Diagrama de Secuencia "Configurar Aplicación" – escenario " Zonas de detalle continuo"**

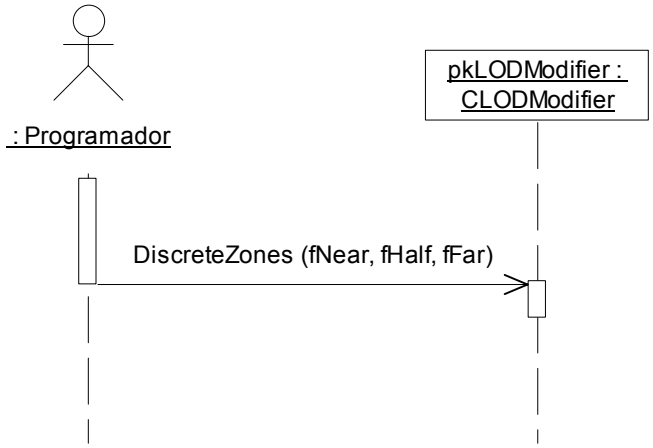



**Fig. 30 Diagrama de Secuencia "Configurar Aplicación" – escenario " Zonas de detalle discreto"**

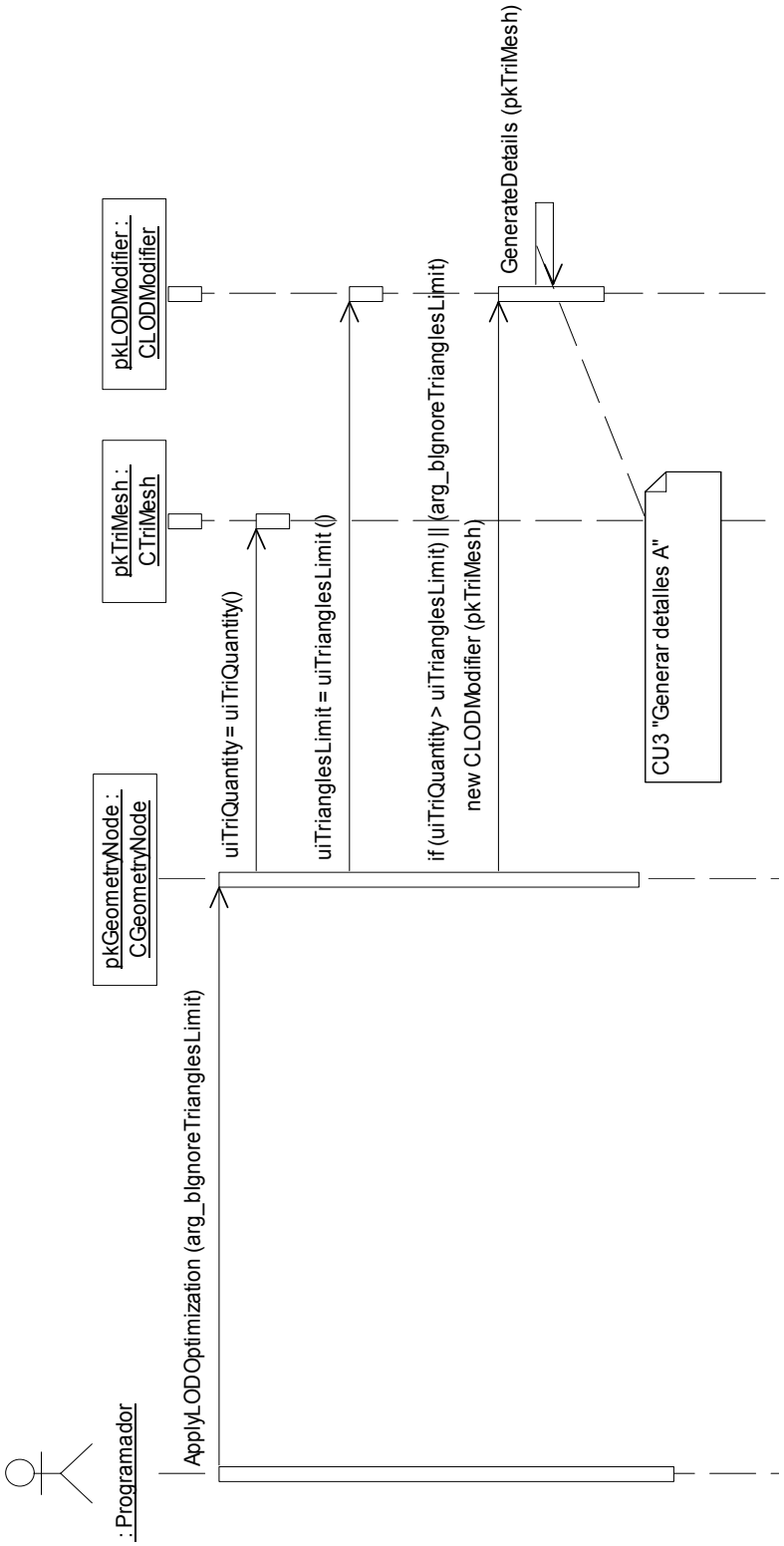



**Fig. 31 Diagrama de Secuencia " Asociar modificador a un objeto"**

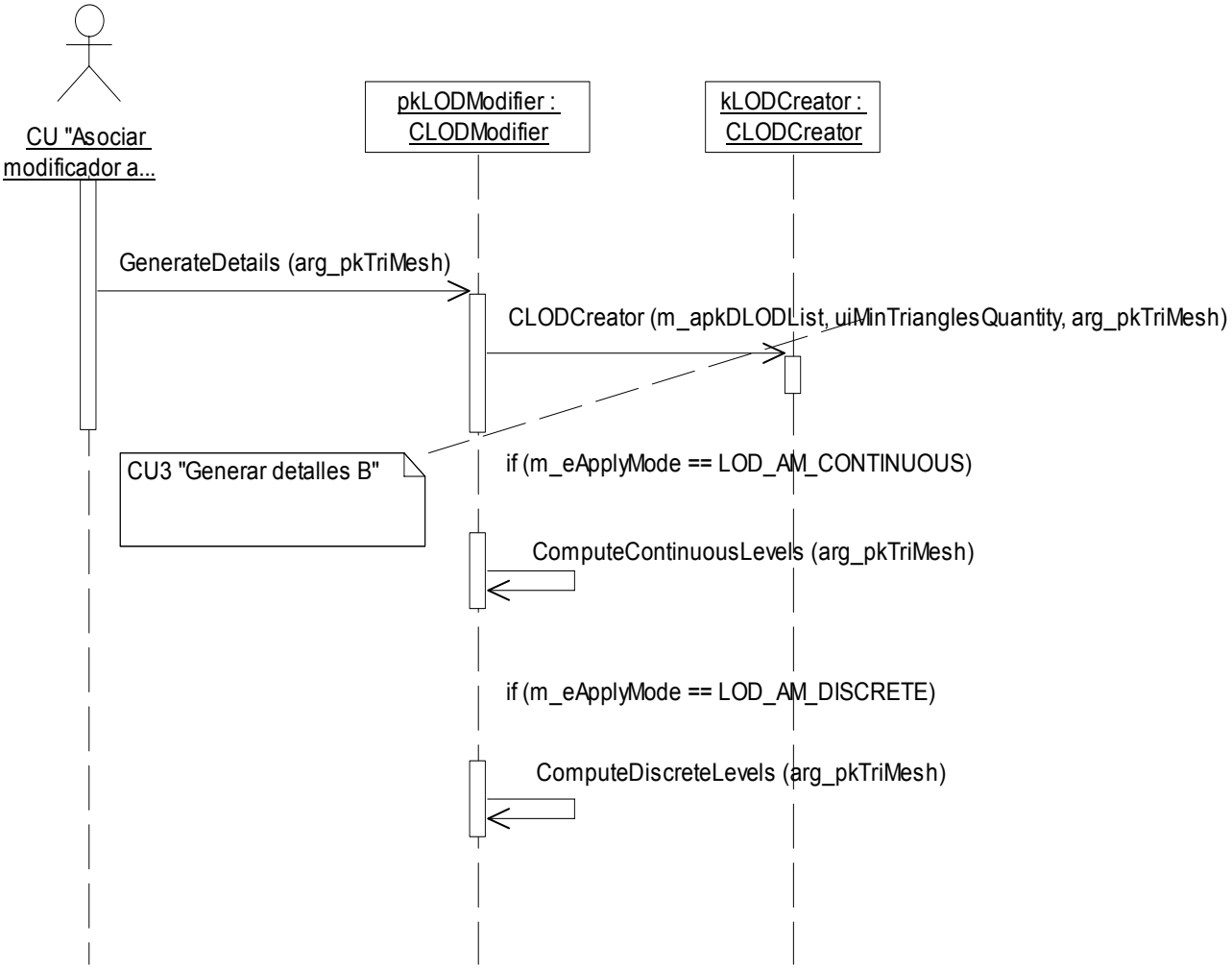



**Fig. 32 Diagrama de Secuencia " Generar detalles A"**

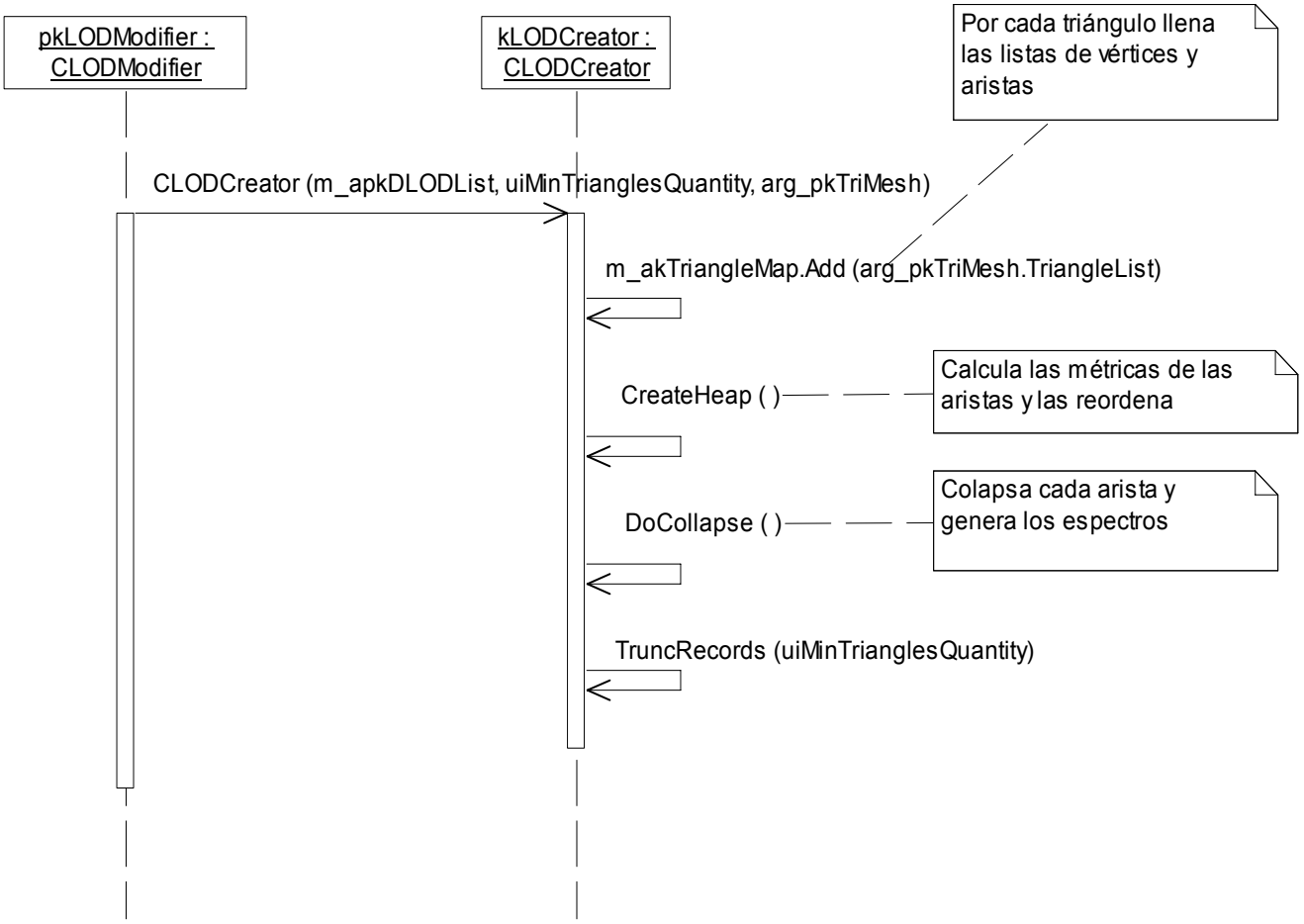



**Fig. 33 Diagrama de Secuencia " Generar detalles B"**

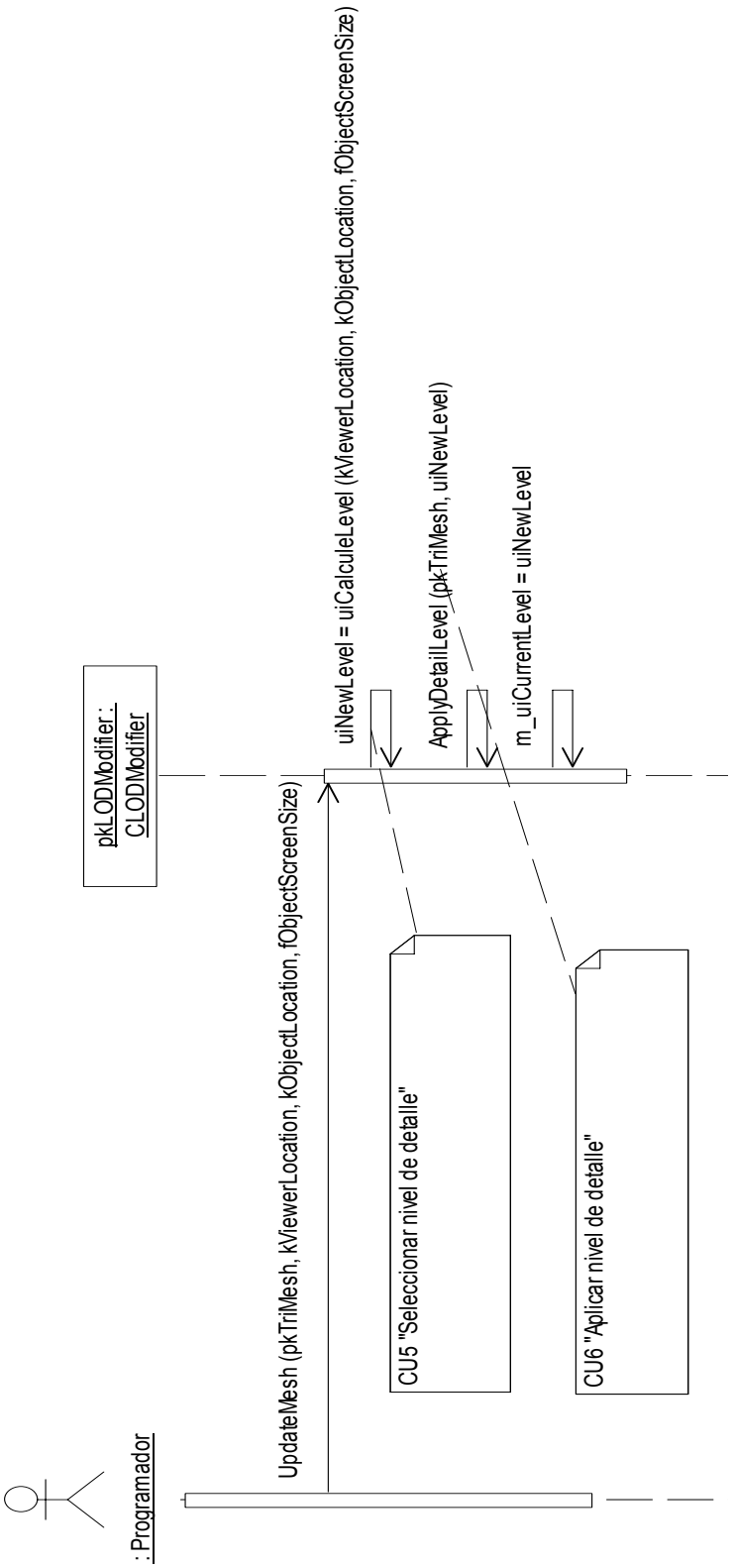



**Fig. 34 Diagrama de Secuencia " Actualizar malla"**

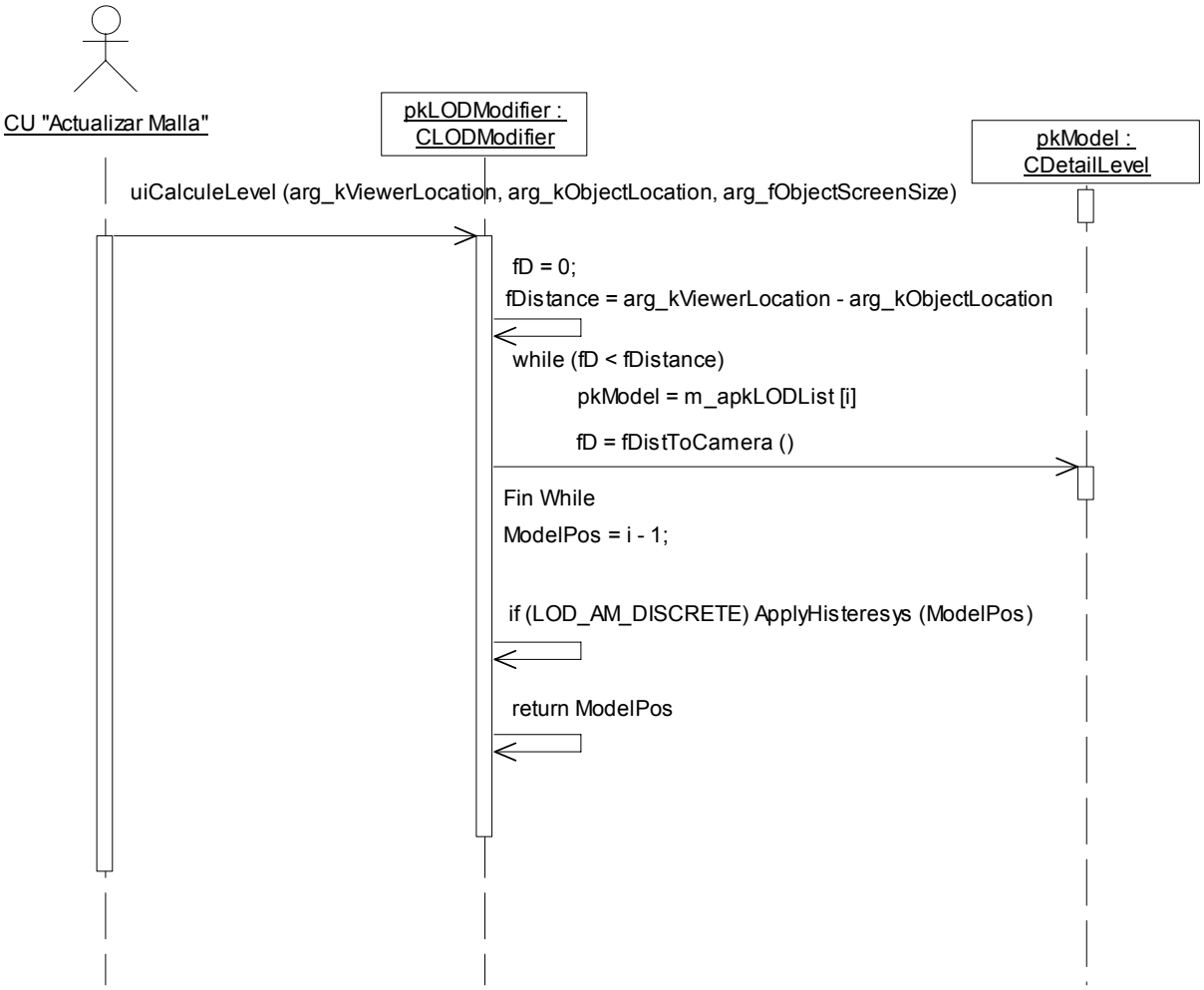



**Fig. 35 Diagrama de Secuencia " Seleccionar nivel de detalle"**

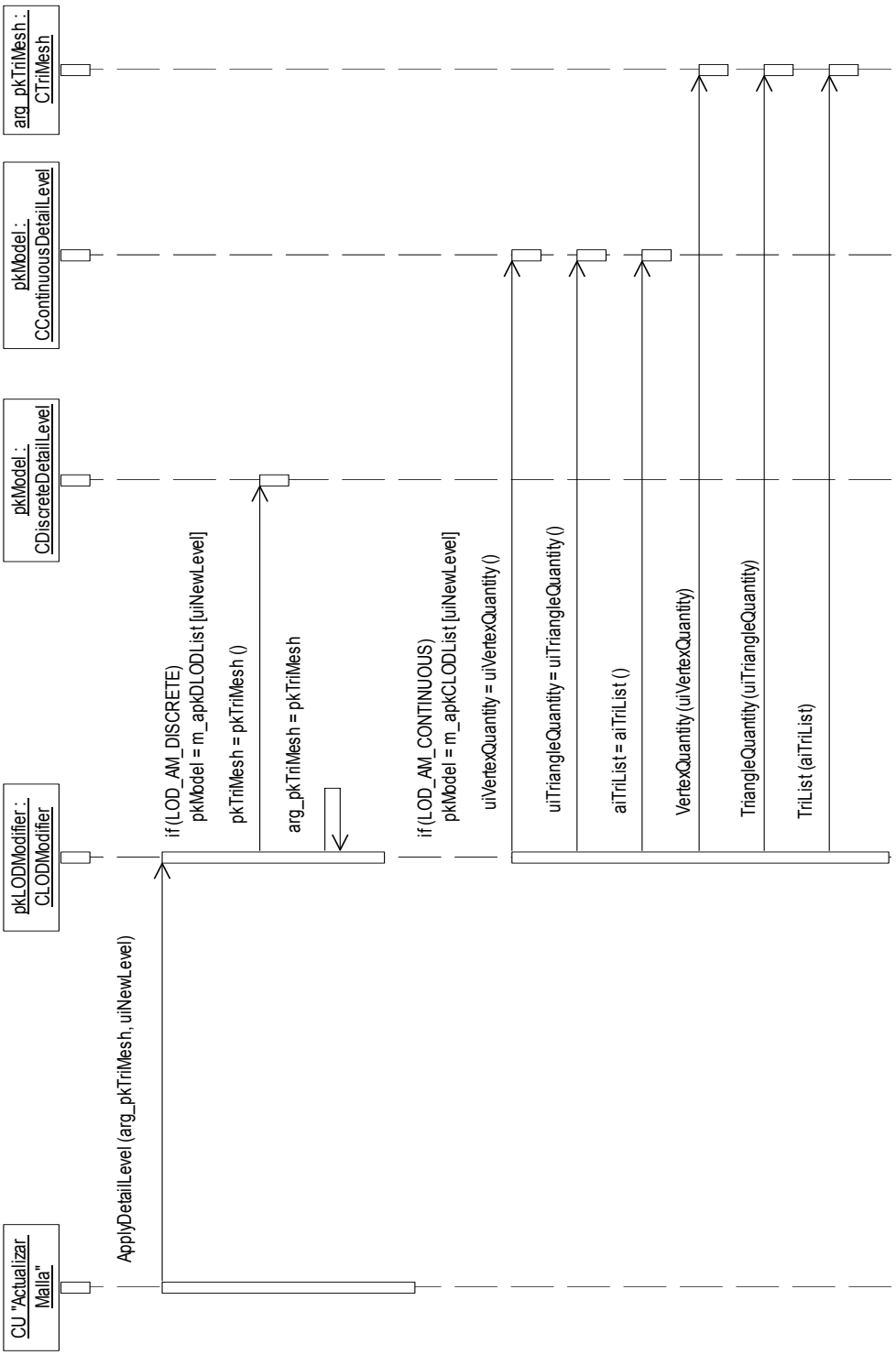



**Fig. 36 Diagrama de Secuencia " Aplicar nivel de detalle"**

## Conclusiones

Como conclusión a este capítulo queda previsto el diseño total del sistema, además las secuencias de pasos para desarrollar los casos de uso traducidas a mensajes entre clases. A partir de estos resultados se puede pasar a la etapa de implementación del proyecto.

.

# 4

# Capítulo 4 Implementación del sistema

## Introducción

A partir de el diseño de clases ya obtenido se crearán componentes físicos, que serán ficheros de extensión .cpp y .h  debido que se implementará haciendo uso del lenguaje C++. Quedará elaborado el diagrama de despliegue.

## 4.1 Estándares de codificación

Esta documentación está propuesta para los usuarios de la biblioteca y programadores de la herramienta. Surge para hacer más entendible la nomenclatura usada en la programación del módulo, permitiendo mayor desenvoltura en el uso de las funcionalidades implementadas. Los programadores harán uso de esta nomenclatura sin alterarla.

Este módulo sigue los estándares del código de la biblioteca a la que se acoplará. Se respetaron los estándares de codificación de C++ y se programó en inglés, dadas las ventajas que ofrecen las características de este idioma.

**Nombre de los ficheros:**

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

Se usará **GT** para identificar el nombre de la herramienta.

**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<> **T**Name**MultiMap**;
TNameMultiMap::iterator **T**Name**MultiMapIter**;

**Declaración de variables:**

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

***Tipos simples:***
bool **b**VarName;
int **i**Name;
unsigned int **ui**Name;

float **f**Name;
char **c**Name;
char* **ac**Name; // arreglo de caracteres
char* **pc**Name; // puntero a un char
char** **aac**Name; // bidimensional
char** **apc**Name; // arreglo de punteros
bool **m_b**MemberVarName; //variable miembro
char **g***c*GlobalVarName; //variable global, no se le antepone “”
short **s**Name;

***Instancias de tipos creados:***
EMyEnumerated **e**Name;
SMyStructure **k**Name;
CClassName **k**ObjectName;
CClassName* **pk**Name; //puntero a objeto
CClassName* **ak**Name; //arreglo de objetos
CClassName* **ak**Name; // variable miembro de clase
IMyInterface* **pI**Name; //puntero interfaces

**Métodos**

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

En el caso de los argumentos se les antepone el prefijo “arg_”

***Constructor y destructor:***

**C**ClassName (bool **arg**_bVarName, float& **arg**_fVarName);
**~C**ClassName ();

***Funciones:***

b*ool bFunction1 (…);*

int* **pi**Function2 (…);

CClassName* **pk**Function3 (…);

***Procedimientos:***

void Procedure4 (…);

**Métodos de acceso a miembros**

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

int **i**MyVar;  //variable

***Obtención del valor:***

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

***Establecimiento del valor:***

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

***Obtención y establecimiento del valor:***

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

## 4.2 Diagrama de despliegue

El diagrama de despliegue es muy simple, incluye solo la representación de una PC, por dicha razón quedó decidido que no es necesario representar dicho diagrama.

## 4.3 Diagrama de componentes

A continuación se representa la relación entre el paquete de componentes STKLODModule y el paquete "STKSceneToolKit", es decir, la herramienta. Esta correspondencia tiene su base en la fuerte relación que existe entre las clases de la herramienta y el módulo de niveles de detalle en desarrollo.

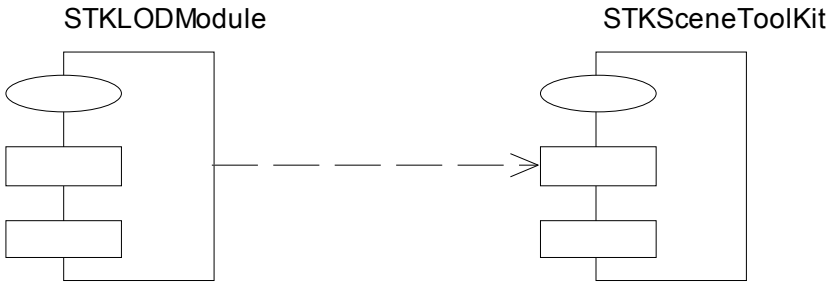



**Fig. 37 Paquetes de componentes**

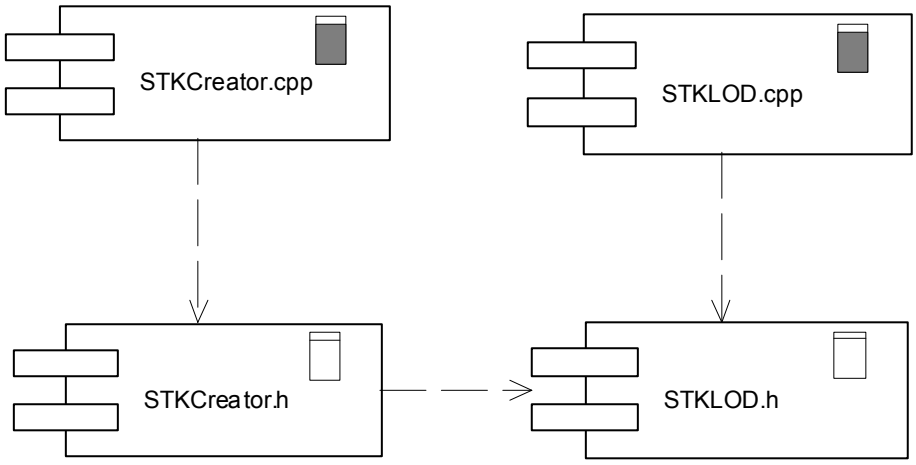



**Fig. 38 Diagrama de Componentes" STKLODModule"**

## Conclusiones

En esta fase, teniendo ya distribuidas las clases por componentes, se encuentra todo listo para la etapa de programación de los casos de uso desarrollados en este primer ciclo. Rational Rose, como herramienta, brinda la posibilidad adicional de generar el código fuente de los componentes que ya han quedado elaborados.

# Conclusiones generales

En función de darle cumplimiento a los objetivos trazados para la realización de este proyecto y en correspondencia con los requerimientos de la entidad cliente, inicialmente se requirió hacer un estudio de la optimización por niveles de detalles, así como algoritmos y tendencias actuales sobre el tema.

En el proceso de estudio se analizaron las principales técnicas de visualización de los motores de Sistemas de Realidad Virtual y los parámetros de los entornos virtuales que se modifican o se optimizan.

Además, a partir de esa investigación se propusieron las características técnicas de la solución.

A continuación se realizó la captura de requisitos funcionales y no funcionales, agrupando los primeros en los casos de uso del sistema. Quedó delimitada la primera fase de desarrollo del proyecto.

Posteriormente se transitó por las etapas de diseño e implementación utilizando los artefactos de RUP, donde surgieron y maduraron las clases, ejecutándose la realización de los casos de uso a desarrollar en la primera fase y se creó el diagrama de componentes final que contendrá a las clases del módulo LOD.

Finalmente, se obtuvo un módulo que responde a todas las necesidades planteadas, logrando una interfaz de programación sencilla con la cual el usuario programador puede optimizar sus aplicaciones finales, sin preocuparse por los detalles internos de procesamiento.

# Recomendaciones

Como recomendaciones para futuras versiones del módulo LOD creado se tienen las siguientes:

- Crear los modelos en un proceso *offline* o de preprocesamiento y no durante la carga de las escenas, y salvar los mismo a un fichero para la carga posterior.
- Incluir la técnica de manejo de niveles de detalle dependiente de la vista del observador.
- Desarrollar la selección del detalle según el área en pantalla que ocupe el objeto.
- Confeccionar un documento siguiendo los estándares de documentación que existen en el proyecto, que funcione como manual de uso y ayuda del módulo.

# Apéndices

## Referencias bibliográficas

### Libros

[1] Clark, J. H. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19:547-554, USA, 1976.

[2] Compilación de autores. *Game Programming Gems.*
Mak DeLoura, USA, 2000
Capítulo 4 “Polygonal Techniques”.
4.9 Yossarian King. “Issues in Geometric Level of Detail Selection”.

[3] LUEBKE David y otros. *Level of Detail for 3D Graphics*
Morgan Kaufmann Publishers, USA, 2003

### Libros digitales

[4] CHOVER, Miguel. *Informática Gráfica*.
http://www3.uji.es/~jromero/grafica (2004).
Capítulo 5. *Visibilidad*.
http://www3.uji.es/~jromero/grafica/Documentos/5-VisibilidadIG35.pdf
Capítulo 9. *Aplicaciones de la informática gráfica*.
http://www3.uji.es/~jromero/grafica/Documentos/9-AplicacionesIG35.pdf

[5] Sense8 CORPORATION TEAM. *WorldToolkit Release 9, Reference Manual.*
Capítulo 4. *Scene Graphs.*
http://www.cs.princeton.edu/courses/archive/spr01/cs598b/papers/wtkr9.pdf (2006)

## Documentos electrónicos

[6] HIDALGO N., José. [Consultado en: 2006]. “Animación en Tiempo Real (Técnicas de Incremento de Velocidad) “.
Disponible en: http://dis.um.es/grupos/sig/08BI/TiempoReal.pdf

[7] KRUS Mike, BOURDOT Patrick Y THIBAULT Guillaume. [Consultado en: 2006]. “Niveles de Detalle y Simplificación Poligonal”.
Disponible en: http://acm.org/crossroads/espanol/xrds3-4/levdet.html

[8] MELERO, F.J., CANO P. y TORRES J.C. [Consultado en: 2007], “Visualización interactiva de SP-Octrees utilizando impostores”.
Disponible en: http://lsi.ugr.es/~fjmelero/invest/ceig04.pdf

## Tesis

[9] PELECHANO GÓMEZ, N. *Estudio y desarrollo de un Sistema de Simulación de Deformaciones Tridimensionales en Figuras Flexibles, orientado a su uso en la Visualización Interactiva de Personajes Virtuales*.
Proyecto de Fin de Carrera, Universidad de Valencia, 2001.

# Bibliografía consultada

1. PARÉS, Narcís. . [Consultado en: 2006]. “Tutorial de VRML97”. Disponible en: http://www.iua.upf.es/~npares/docencia/vrml/tutorial_e.htm
2. GONZÁLEZ CORDERO, Carlos L. [Consultado en: 2006]. “Tutorial de VRML. Tema 14: Niveles de detalles”. Disponible en: http://wwwdi.ujaen.es/~rsegura/igai/vrml/documentos/tema14.htm
3. LINDSTROM Peter, KOLLER David. [Consultado en: 2006]. “Real-Time, Continuous Level of Detail Rendering of Height Fields”. Disponible en: http://www.cc.gatech.edu/gvu/people/peter.lindstrom/papers/siggraph96.pdf
4. ULRICH, Thatcher. [Consultado en: 2007]. “Continuous LOD Terrain Meshing Using Adaptive Quadtrees”. Disponible en: http://www.gamasutra.com/features/20000228/ulrich_01.htm
5. ROETTGER Stefan, HEIDRICH Wolfgang, SLUSALLEK Philipp, y SEIDEL Hans-Peter. *Real-Time Generation of Continuous Levels of Detail for Height Fields*. [en línea] In V. Skala, editor, Proc. WSCG '98, pág. 315-322, 1998. [Consultado en: 2007]. Disponible en: http://stereofx.org/aboutme.html#Papers
6. SUÁREZ SÁNCHEZ, Fernando. [Consultado en: 2006]. “Compensación de movimiento: interpolación por mallas de control”. Disponible en: http://www.gti.ssr.upm.es/gente/ex/pfcs/pfces/fss/compensacion3.html
7. WIKIPEDIA. [Consultado en: 2006] “Morphing”. Disponible en: http://en.wikipedia.org/wiki/Morphing

# Glosario de abreviaturas

**CAD:** Computer-Aided Design (Diseño asistido por computadora).

**FPS:** Frames per second (fotogramas por segundo).

**LOD:** Level of Detail (Nivel de detalle).

**NURBS:** Non Uniform Rational B-splines (B-splines racionales no uniformes).

**PTM:** Projective texture mapping (mapeo proyectivo de texturas).

**RV:** Realidad Virtual.

**SRV:** Sistema de Realidad Virtual.

**UCI:** Universidad de las Ciencias Informáticas.

# Glosario de términos

**A**

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

***Anisotrópico:*** dependiente de la vista.

***Aristas no múltiples (no manifold):*** aristas con uno, tres, o más triángulos adyacentes.

**C**

***Clipping:*** recorte de un triángulo cuando se realiza la proyección 3D o 2D de los vértices.

**E**

***Engine:*** programa orientado a la creación de otros softwares.

**F**

***Face clustering:*** agrupamiento de caras.

***Fotograma:*** cada uno de las imágenes que componen una animación.

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

***Frustum:*** eliminación de los objetos fuera del campo de visión de la cámara.

***Frustum Culling:*** selección por prisma o cono de visión.

**G**

***Género de la malla:*** el número de agujeros en la superficie de la malla.

***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.

**H**

***Hysteresis thresholding (histéresis de umbral):*** un retardo en la transición de niveles de detalles, en la cual los objetos pasan con mayor suavidad de un nivel a otro.

**I**

***Isosuperficies:*** formas descritas por funciones matemáticas. Superficies equipotenciales, útiles para representar formas suaves: orgánicas y moleculares.

***Isotrópico:*** independiente de la vista.

**L**

***Level Of Detail (Nivel de Detalle):*** un modelo con una resolución particular entre una serie de modelos del mismo objeto. El mejor resultado gráfico que se puede obtener es usando un menor LOD cuando el objeto ocupa menos píxeles en la pantalla.

***LOD selection (selección del detalle):*** es uno de los pasos de la tubería de visualización donde se tienen los objetos realmente visibles, y se trata de optimizarlos escogiendo el nivel de detalle más adecuado para cada uno.

**M**

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

***Manifold:*** múltiple.

***Motores de SRV:*** armazón orientada a objeto con las funcionalidades básicas para construir Sistemas de Realidad Virtual.

**O**

***Oclussion Culling (selección por oclusión):*** uno de los pasos de la tubería de visualización elimina los objetos ocultos por otros objetos.

**P**

***Popping (back and forth popping):*** salto alante y atrás. Efecto que ocurre cuando un objeto constantemente salta entre dos niveles, por ejemplo, al moverse en zigzag por la zona de cambio del nivel de detalle.

***Projective texture mapping:*** proyección de una textura.

**R**

***Rastering:*** rellenado de los triángulos durante el proceso de *rendering*.

***Render:*** componente del sistema gráfico de la computadora, responsable de dibujar las geometrías de un modelo utilizando la información de estados de dibujo, llamada estados de *render*.

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

**S**

***Splines:*** curva definida a trozos mediante polinomios. La simplicidad de la representación y la facilidad de cómputo de los *splines* los hacen populares para la representación de curvas en informática, particularmente en el terreno de los gráficos por ordenador.

**T**

***Tiempo real:*** define un tiempo de respuesta, de algún sistema muy rápido, dando la apariencia de que la respuesta del sistema ocurre al mismo tiempo que éste es excitado.

***Topología de la malla:*** conectividad de la malla, representada por los bordes y caras que conectan a los vértices, el número de huecos, túneles y cavidades en la malla.

**V**

***Virtuoterapia:*** empleo de la Realidad Virtual para curar fobias y traumas.

***Visibility pipeline (tubería de visualización):*** una serie de pasos que se realizan para lograr una visualización eficiente en SRV.

***Voxels:*** volumen cúbico de píxeles para cuantizar un espacio tridimensional.