Memoria de Tesis Doctoral

SP-OCTREE
Representación Jerárquica de Sólidos Poliédricos

Doctorando

Pedro Cano Olivares

Director

Juan Carlos Torres Cantero

Departamento de Lenguajes y Sistemas Informáticos
Escuela Técnica Superior de Ingeniería Informática
Universidad de Granada

Universidad de Granada

E.T.S. de Ingeniería Informática

Departamento de Lenguajes y Sistemas Informáticos

SP-OCTREE: REPRESENTACION JERARQUICA DE SOLIDOS POLIEDRICOS

Memoria presentada por:

Pedro Cano Olivares

para la obtención del grado de Doctor en Informática

Director

Dr. D. Juan Carlos Torres Cantero

Granada, Marzo de 2004

La memoria titulada "***SP-OCTREE*: REPRESENTACIÓN JERÁRQUICA DE SÓLIDOS POLIÉDRICOS"**, que presenta *D. Pedro Cano Olivares* (miembro del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada) para optar al grado de Doctor en Informática, ha sido realizada en el Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada bajo la dirección del Doctor *D. Juan Carlos Torres Cantero*.

Granada, Marzo de 2004.

El Doctorando — El Director

Pedro Cano Olivares — Juan Carlos Torres Cantero

Para *Rafi*, *Pedro* y *Rafa*,
porque todo el tiempo empleado en este
trabajo debería habérselo dedicado a ellos.

Agradecimientos

Han sido muchos años de esfuerzo y dedicación hasta que éste trabajo se ha completado, y son muchas las personas que de alguna u otra forma han influido en el desarrollo del mismo. Por ello, son muchos los agradecimientos que tengo que dar.

A mis padres, porque gracias a ellos y a su esfuerzo he podido llegar donde estoy. Sin ellos nada de esto hubiera ocurrido.

A mis amigos y compañeros de despacho, Jorge y Marcelino, por todos los buenos ratos que hemos pasado y que seguro pasaremos juntos.

A todos mis compañeros del Departamento de Lenguajes y Sistemas Informáticos, y en especial a Paco, por sus útiles comentarios desde que comenzamos a trabajar en estos temas, y a Javi, por soportar este trabajo e intentar continuarlo.

A Rafa Segura, por facilitarme algunos de los algoritmos utilizados para las evaluaciones. A Isabel Navazo y Eva Monclús, de la UPC, por ofrecerme algunos de los modelos de sólidos utilizados en las pruebas y la implementación de los Octrees Extendidos.

A Juan Carlos, por aconsejarme quedarme en Granada, por ser un ejemplo de lo que me gustaría llegar a ser y por todo el tiempo y esfuerzo que ha dedicado en la dirección de este trabajo.

Y, por supuesto, a quien más se lo merece. A mis hijos Pedro y Rafa, por ser la alegría que ha llenado mi vida, y a mi mujer, Rafi, por ser como es, por darme todo su apoyo aún cuando he fallado y compartir todos los momentos, buenos y malos, de nuestra vida juntos. Intentaré compensar el tiempo que no os he podido dedicar durante la realización de este trabajo.

El trabajo presentado en esta memoria ha sido parcialmente subvencionado por el *Ministerio de Ciencia y Tecnología* y fondos *FEDER* a través del proyecto de investigación TIC2001-2099-C03-02.

Contenido

Lista de Tablas

Lista de Figuras

Lista de Algoritmos

CAPÍTULO

1

Introducción

1

En este capítulo realizaremos una pequeña introducción al trabajo que presentamos en esta memoria. Comentaremos las motivaciones que nos hicieron comenzar a trabajar sobre el tema en cuestión, describiremos los objetivos básicos planteados y presentaremos las aportaciones más importantes obtenidas que detallaremos en los distintos apartados en que hemos estructurado la memoria.

1.1. Presentación de la Tesis

El tema principal del trabajo realizado para escribir esta memoria se encuadra dentro del *modelado de sólidos*, campo de la informática que surge por la necesidad de representar en el ordenador modelos de objetos para poder automatizar determinadas tareas sobre los mismos.

Podemos definir un *modelo de un sólido* como una descripción de un subconjunto de los atributos o relaciones que lo definen.

Estas representaciones permiten trabajar con ayuda del ordenador sobre los modelos, como si lo hiciéramos sobre los objetos que representan, para realizar simulaciones o calcular características de estos últimos sin tener que actuar sobre ellos directamente.

La utilización de estos modelos ha sido una de las bases para la aparición de dos procesos que en la actualidad tienen gran importancia: el Diseño Asistido por Ordenador (CAD) y la Fabricación Asistida por Ordenador (CAM).

1.1.1. Motivación

Existen varios esquemas de representación de sólidos, cada uno con sus ventajas e inconvenientes, entre los que podemos encontrar algunos basados en la descomposición del espacio ocupado por el sólido, y que utilizan una representación jerárquica como forma de almacenamiento de la información que define el modelo.

Estos esquemas hacen muy simples determinadas operaciones sobre el modelo, pero presentan el inconveniente de ser tan sólo una aproximación del mismo, además de necesitar gran cantidad de recursos de almacenamiento.

Se han propuesto distintas extensiones a estos modelos con el fin de poder representar un número mayor de objetos de forma exacta y con menor necesidad de recursos, manteniendo las características de los métodos jerárquicos originales.

Tras realizar un estudio en profundidad de estos esquemas de representación de sólidos basados en métodos jerárquicos, el trabajo presentado en esta memoria se centra en el diseño e implementación de un nuevo esquema de representación que, conservando las propiedades de direccionabilidad y compactación de los esquemas originales, permite representar de forma exacta sólidos poliédricos, con menor necesidad de recursos y que mejora algunas de las características o procesos a realizar con el modelo.

Para llevar a cabo esta idea nos planteamos la posibilidad de extender los métodos originales mediante la modificación de la información del sólido almacenada en la estructura que define el modelo.

1.1.2. Objetivos

A partir del estudio de los métodos jerárquicos existentes y de las características de los mismos, los objetivos básicos que nos planteamos en este trabajo fueron:

1. Obtener un nuevo esquema de representación de sólidos basado en esos métodos jerárquicos que mejore determinados aspectos respecto a los esquemas existentes.
2. Implementar las operaciones básicas sobre sólidos dentro del nuevo esquema.

3. Implementar operaciones booleanas con modelos representados con el esquema propuesto.
4. Implementar algoritmos de visualización del modelo, tanto de forma directa como de forma adaptativa.
5. Comparar el esquema propuesto con otros esquemas de representación en cuanto al espacio necesario y en cuanto a complejidad en las operaciones.
6. Utilizar el esquema para transmisión progresiva de modelos utilizando las posibilidades de visualización adaptativa que ofrece.

Toda la implementación del esquema se debía realizar sobre una arquitectura para modelado y visualización de sólidos y volúmenes diseñada dentro del grupo de Informática Gráfica de la Universidad de Granada, lo que va a permitir de forma simple la comparación con otros esquemas implementados sobre la misma.

1.2. Organización de la memoria

En la presente memoria hemos tratado de resumir de forma clara y concisa el trabajo realizado durante los últimos años y los resultados obtenidos a partir del mismo. Algunos de los trabajos que presentamos aquí ya han sido publicados o presentados en revistas y congresos nacionales e internacionales.

En el capítulo 2 se presentarán los conceptos básicos del modelado de sólidos, línea en la que se encuadra el trabajo realizado. Se realizará un recorrido por los distintos esquemas de representación de sólidos, centrándonos especialmente en aquellos que han servido de base para el trabajo que aquí presentamos.

En el capítulo 3 se propone el nuevo esquema de representación de sólidos basado en los Octrees clásicos, en las extensiones propuestas de estos y en los árboles BSP.

A este nuevo esquema lo hemos denominado ***SP-Octrees*** *(Space Partition Octrees)* debido a que en los nodos del árbol realizamos una descomposición

espacial a partir de la información de la frontera del sólido que aparece en los mismos. En este mismo capítulo se discute la representación interna que hemos utilizado para almacenar un modelo siguiendo este nuevo esquema de representación, así como las características principales del mismo.

En el capítulo 4 se estudian las operaciones básicas sobre modelos sólidos representados con el esquema propuesto. Se describirá un algoritmo para la clasificación de elementos geométricos respecto al modelo, se presentará el proceso utilizado para obtener una visualización del sólido representado mediante el esquema y se describirá el proceso de reconstrucción de la frontera del objeto representado.

En este capítulo se describe también el proceso de construcción de un modelo en el esquema propuesto a partir de algunos de los esquemas de modelado de sólidos más utilizados, así como el proceso inverso por el cual convertimos un modelo en nuestro esquema al equivalente en otros esquemas. Después se realizará un estudio de los algoritmos propuestos y una comparación con otros esquemas de representación existentes.

Otras de las operaciones más importantes en un esquema de representación de sólidos son las operaciones booleanas, que permiten diseñar nuevos objetos a partir de combinaciones de otros. En el capítulo 5 se describen esas operaciones sobre el nuevo esquema de representación propuesto.

Por último, en el capítulo 6 se describe un mecanismo de transmisión progresiva de modelos sólidos representados con el esquema que proponemos. Este proceso permite al receptor tener un modelo válido del sólido representado desde que comienza a realizarse la transmisión desde el emisor, y en sucesivas fases del proceso transmitir representaciones válidas del mismo a diferente nivel de detalle.

Finalizaremos la memoria con el capítulo 7, resumiendo las principales aportaciones que se extraen del trabajo realizado y presentando las líneas que quedan aún sin concluir y que serán los temas en los que trabajaremos a partir de ahora.

CAPÍTULO

2

Modelado de Sólidos

2

En este capítulo haremos en un repaso general de algunos de los conceptos básicos del campo en el que nos movemos en nuestro trabajo. Presentaremos una definición clásica del modelado de sólidos y del proceso de representación de esos modelos. Describiremos los esquemas de representación más utilizados, prestando atención a las características más importantes de cada uno de ellos, y centrándonos en aquellos que han servido como base al trabajo realizado.

2.1. Introducción

Podemos entender el ***modelado de sólidos*** como un conjunto de herramientas y técnicas orientadas a la representación de objetos sólidos. Esa representación permite a partir de un objeto, existente o no en la realidad, obtener un modelo del mismo utilizando una determinada estructura de datos dentro del ordenador.

En muchas aplicaciones se hace necesario realizar determinadas operaciones con objetos reales tales como:

- Distinguir entre interior, exterior y la superficie del objeto.
- Realizar operaciones entre dos objetos distintos.
- Visualizarlo desde un determinado punto de vista.
- Obtener propiedades del objeto:
  - Calcular volumen, peso, etc.
  - Estudiar su respuesta a factores externos (resistencia, flexibilidad, etc.)

Las propiedades a calcular, generalmente, dependen del tipo de aplicación (seguridad, aerodinámica, térmicas). Muchos de estos problemas al final se basan en aspectos geométricos. Por ello, un modelador de sólidos debe describir adecuadamente objetos sólidos y ofrecer un repertorio de algoritmos que operen sobre la geometría representada para poder realizar cálculos sobre el modelo.

El modelado de sólidos ha sido un campo de la informática muy estudiado [Baer79, Bron87, Brun92, Hoff89, Mänt88, Mänt89, Mort85, Requ80, Requ82, Requ83, Roos99].

Requicha lo definió [Requ83] como: "el conjunto de teorías, técnicas y sistemas orientados a la representación 'completa en cuanto a información' de sólidos. Dicha representación debe permitir (al menos en principio) calcular automáticamente cualquier propiedad bien conocida de cualquier sólido almacenado".

Para Mortenson [Mort85] un modelo geométrico de un sólido es una "representación matemática, no ambigua, e informacionalmente completa, de la forma de un objeto físico tal que dicha representación pueda ser procesada por un ordenador".

Como a veces resulta difícil y complejo partir del objeto real para su análisis y simulación, deberemos construir un modelo del sólido para simular su comportamiento.

Podemos enfocar el análisis del problema de construcción de un modelo desde 3 niveles distintos de abstracción (ver figura 2.1) [Mänt88, Requi80]:

1. **Objetos físicos**: No podemos percibir un objeto real con toda su complejidad, y mucho menos representar todos sus aspectos sobre un ordenador, para poder hacer cálculos sobre sus propiedades.

2. **Objetos matemáticos**: Son una idealización adecuada del objeto real. Un modelo matemático de un objeto real es una abstracción de las características del objeto que permite simular su comportamiento bajo un conjunto de hipótesis. Permiten establecer una conexión simple con el mundo real para su manipulación.

3. **Representación**: Una vez establecido el modelo matemático, debemos asignarle una representación adecuada para el ordenador.

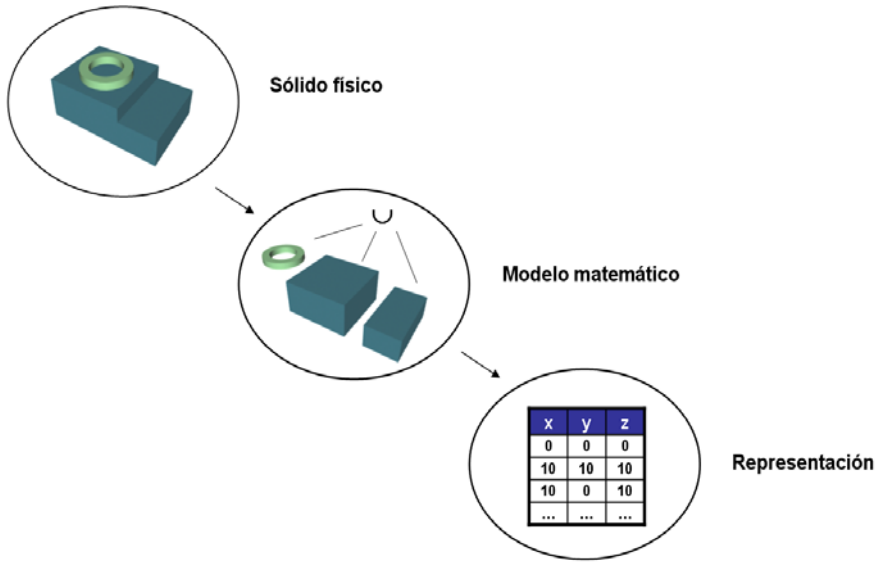


Figura 2.1. Niveles de abstracción en el proceso de modelado de sólidos.

2.2. Modelo matemático

Un primer problema a resolver es determinar el concepto de sólido. Podemos caracterizar el modelo matemático de un sólido mediante dos aproximaciones distintas [Mänt88]:

- objeto sólido en el espacio euclídeo (*topología puntual*), u
- objeto definido por su contorno (*topología algebraica*).

2.2.1. Topología puntual (conjunto de puntos)

Según esta aproximación un sólido es un subconjunto *S*, cerrado y acotado, del espacio euclídeo E^3 [Mänt88].

Pero para que un subconjunto de puntos cerrado y acotado sea un sólido, debe satisfacer además los siguientes requerimientos [Requ80]:

a) *Rigidez*: un sólido debe ser invariante ante transformaciones rígidas (traslaciones y rotaciones).

b) *Regularidad*: todo el sólido debe estar compuesto por materia, sin existir puntos, aristas o caras aisladas.

   Esta característica se puede formalizar a partir del concepto de *regularización de un conjunto* de puntos *A*, *r(A)*, que se define como:

$$r(A) = \text{clausura (interior (A))}$$
$$\text{clausura (A)} = (A \cup b(A))$$

   donde *b(A)* es el conjunto de puntos frontera. Decimos que un conjunto *A* es ***regular*** si $A \equiv clausura(A)$. A un conjunto regular y acotado se le denomina ***r-set***.

c) *Representación finita:* si queremos almacenar y tratar en un ordenador un conjunto continuo de puntos, debemos tener una representación finita del mismo.

2.2.2. Topología algebraica

Con esta otra aproximación se intenta caracterizar el sólido mediante información de la superficie del objeto [Mänt88]. Este modelo asume que el sólido tiene un interior homogéneo y no se almacena información del mismo, sino que sólo se representa la superficie que lo delimita del exterior.

Pero debemos tener en cuenta que, al igual que hay conjuntos de puntos que no forman un sólido válido, para representar un sólido con esta aproximación la superficie debe formar una piel completa y cerrada del objeto, y no debe intersecar consigo misma, delimitando así perfectamente el interior del exterior del sólido (la superficie debe ser orientable). A esta superficie se denomina *frontera* del sólido y formalmente se define como el espacio topológico *2-variedad* [Mänt88].

De esta forma realizamos una definición matemática del sólido en función de la superficie (2D) en vez de su volumen (3D). Esta representación suele ser muy utilizada para almacenar un modelo en el ordenador [Kauf94] por facilitar el proceso de visualización y procesamiento del sólido.

2.2.3. Definición de sólido

Podemos recoger ambas aproximaciones y reflejar las características que debe cumplir un objeto sólido. De esta forma, podemos definir un ***sólido*** [Requ80] como un subconjunto del espacio euclídeo E^3 que cumple las siguientes características:

1. *Rigidez*: el sólido tiene una forma fija e invariante que es independiente de su posición y orientación.
2. *Homogeneidad*: todo el sólido está compuesto por materia y no debe presentar puntos, aristas o caras aisladas, es decir, debe ser un conjunto *regular*.
3. *Finitud*: debe ocupar una porción finita del espacio.
4. *Describible finitamente*: debe de existir algún aspecto del sólido que asegure el que se pueda representar como una secuencia finita de datos.
5. *Invariante* respecto a transformaciones geométricas.
6. *Determinado por el contorno:* debe existir una frontera que delimite el interior del exterior del sólido.

2.2.4. Operaciones regularizadas

A la hora de combinar objetos mediante operaciones booleanas, puede haber casos donde el resultado no satisfaga la condición de regularidad. Por ello, es necesario usar las *operaciones booleanas regularizadas* ($\cup^*, \cap^*, -^*$), que garantizan que los resultados de las operaciones sobre sólidos siguen siendo sólidos [Fole96].

Intuitivamente, el sólido resultante no contendrá puntos de la frontera si no hay adyacencias a puntos interiores del objeto resultante (ver figura 2.2).

Con las definiciones descritas en el apartado 2.2.1, podemos obtener las operaciones booleanas regularizadas del siguiente modo:

$$A\ op^*\ B = clausura\ (interior\ (A\ op\ B))$$

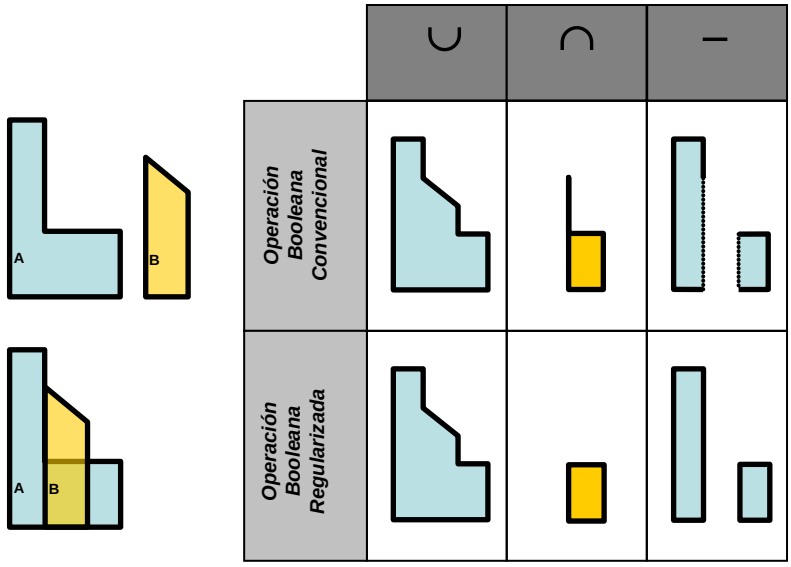


Figura 2.2. Operaciones Booleanas Regularizadas.

2.3. Esquemas de representación

Una vez definido el modelo matemático del sólido, nos interesa tenerlo representado con una estructura de datos para que sea manipulable en el ordenador.

Un ***método de representación de sólidos*** se puede entender como una relación entre sólidos abstractos y descripciones de éstos.

Una ***representación sintácticamente correcta de un sólido*** es una colección finita de símbolos (de un alfabeto finito) que describen un sólido siguiendo una serie de reglas sintácticas de un determinado esquema de representación [Mänt88].

Formalmente un ***esquema de representación S*** se define como una aplicación del *espacio de modelos matemáticos* ***M*** (compuesto por sólidos abstractos) en el conjunto de todas las representaciones sintácticamente correctas o *espacio de representación* ***R*** (figura 2.3).

2.3.1. Propiedades de los esquemas de representación

Cada esquema de representación se puede caracterizar a través de las siguientes propiedades formales [Requ80]:

(1) **Dominio**: Caracteriza el poder descriptivo del esquema de representación. Es el conjunto de sólidos $\boldsymbol{D \subseteq M}$ que se puede modelar con él.

Lo ideal sería tener un método cuyo dominio sea todos los sólidos, pero eso no es posible, sino que un método servirá para determinados sólidos dependiendo de la aplicación (*dominio ≠ espacio de modelos matemáticos*).

Además, debemos tener en cuenta que no todos los elementos del conjunto de representaciones tienen por qué ser imagen de algo (*conjunto imagen ≠ espacio de representación*), es decir, usados para modelar. Esto hace aparecer el concepto de validez.

(2) **Validez**: Un método de representación es válido si el conjunto imagen coincide con el conjunto de representaciones, es decir, cualquier representación es generada por algún sólido con ese método.

Una representación es *no valida* si no corresponde a ningún sólido real. El conjunto de todas las representaciones válidas será el *rango del sistema* $\boldsymbol{V}$.

Es importante que el método de representación asegure de por sí la validez de todas las representaciones que se pueden crear o que existan procedimientos automáticos para comprobar la validez de las representaciones.

(3) **Completitud o no ambigüedad**: Un método es no ambiguo si cada representación se corresponde con un único objeto real. Para poder comprobar la igualdad de objetos dentro del esquema es necesario que el método sea no ambiguo.

(4) **Unicidad**: Se dice que existe unicidad en el método de representación si cada objeto real tiene, a lo sumo, una posible representación. Al igual que ocurre con la ambigüedad, si el método no asegura la unicidad, no es posible comprobar la igualdad entre objetos.

Lo ideal sería un método de modelado con dominio amplio, válido, que asegure la unicidad y no ambiguo, pero no todos los esquemas de representación cumplen todas estas características.

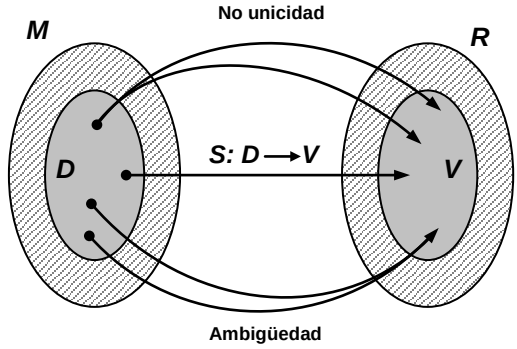


Figura 2.3. Esquema de representación *S*.

Además, para poder usar de forma adecuada y eficiente un esquema de representación, este debe poseer una serie de propiedades importantes a nivel práctico para su implementación:

(5) Concisión: el espacio ocupado por una representación ha de ser el menor posible, conteniendo pocos datos redundantes. Aunque a veces, una redundancia controlada puede tener ventajas al reducir el tiempo de procesamiento.

(6) Facilidad de creación y edición: la facilidad con que se crean las representaciones es muy importante. Son más fáciles de editar las representaciones concisas ya que si el esquema se basa en representaciones detalladas deberá de existir un subsistema de entrada que facilite su edición.

(7) Facilidad de visualización y cálculo.

(8) Eficiencia en el contexto de la aplicación: cada esquema se utiliza en campos concretos, por lo que es deseable que sea eficaz en ese campo.

La existencia de representaciones múltiples introduce problemas de redundancia en sistemas de modelado, provocando problemas de consistencia, por lo que deberemos asegurar que varias estructuras que pretenden representar el mismo objeto en diferentes situaciones no aporten informaciones contradictorias.

2.3.2. Esquemas de representación de sólidos

Como hemos descrito en el apartado anterior, cada método de representación presenta una serie de propiedades que lo caracterizan. Revisaremos a continuación algunos de los esquemas de representación más utilizados.

a) Instanciación

Los sólidos se describen como un conjunto de primitivas parametrizadas predefinidas. Modificando los parámetros podemos modelar nuevos objetos (figura 2.4).

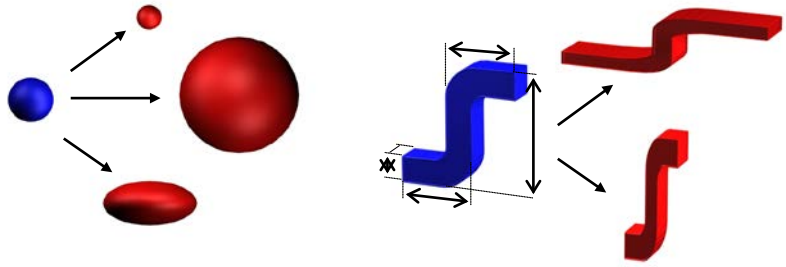


Figura 2.4. Instanciación.

Cada objeto y el cálculo de sus propiedades se describen mediante procedimientos propios a cada primitiva en función de los parámetros que lo definen.

No existen mecanismos de combinación de objetos, por lo que su dominio es muy limitado al depender del tipo de primitivas de partida.

La validez del modelo viene dada por limitaciones que se establecen en los parámetros. Es un método no ambiguo, pero no único.

b) Barrido

Se suele utilizar para generar objetos cuya sección es constante y suele encontrarse como método de construcción en sistemas que usan otros esquemas de representación.

Aunque su representación interna es muy concisa, el dominio de objetos que pueden ser representados es muy limitado.

Un objeto generado por barrido se descompone en un objeto rígido más simple al que normalmente se denomina *objeto generador* (que suele ser una superficie) y una trayectoria llamada *objeto director* (una curva en el espacio).

El sólido se define por el volumen barrido por la superficie al desplazarse por la trayectoria [Bron88]. Ese barrido puede ser *traslacional* (conocido también como *extrusión*) donde la trayectoria es una línea recta, o *rotacional* donde la trayectoria es circular (figura 2.5).

También podemos definir un barrido más general, donde el generador no tiene por qué ser rígido, y tanto este como la trayectoria pueden ser de cualquier tipo.

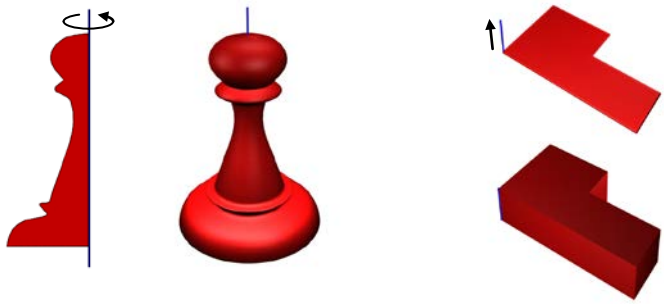


Figura 2.5. Representación por barrido rotacional y trasnacional.

Como vemos, estos dos esquemas presentan un dominio de representación muy limitado o necesita un gran número de primitivas para ser útil. Además, están muy orientados a determinado tipo de objetos, por lo que son necesarios esquemas de representación más genéricos. Estos los podemos dividir en tres grandes grupos:

c) Modelos de fronteras

En este esquema se considera el sólido como un volumen delimitado por una superficie cerrada. Esa superficie determina qué es interior y qué es exterior al sólido, la apariencia del objeto, y además es el medio de contacto con otros sólidos.

Esta frontera se descompone en una serie de *caras*, cada una de las cuales se describe por la frontera que separa los puntos interiores y exteriores de la misma [Baer79].

Se debe hacer una descripción tanto de la geometría como de la topología del objeto. Por ello, la representación normalmente utilizada está basada en grafos que permiten describir las relaciones entre las distintas partes de la frontera.

d) Modelos constructivos

Partimos de un conjunto de sólidos simples que se definen geométricamente a partir de parámetros. Sobre estos objetos se definen operaciones booleanas (unión, intersección y diferencia) y sobre esta base, un sólido cualquiera se representa como una combinación de objetos primarios mediante operaciones booleanas.

Normalmente se representa utilizando árboles binarios en los que en los nodos terminales tenemos las *primitivas* y en los nodos internos tenemos operaciones booleanas y transformaciones.

e) Modelos de descomposición espacial

Estos esquemas se basan en la representación de un sólido como la suma o unión de un conjunto de *celdas* simples, disjuntas y adyacentes, en las cuales está dividido. Con esto se consigue tratar de forma sencilla un sólido que en conjunto puede ser difícil de representar.

En los siguientes apartados veremos con más detalle algunos de los esquemas de representación de sólidos más representativos de estos grupos y que, de alguna forma, han servido como base de trabajo en el proceso de diseño del nuevo esquema de representación propuesto en esta memoria.

2.3.3. Modelos de fronteras (B-rep)

Es uno de los esquemas de representación más utilizados en los sistemas de modelado actuales. Se basa en la observación de que todo objeto sólido está limitado por una frontera compuesta por un conjunto de caras, definidas a partir de aristas y vértices [Baer79]. Esta frontera separa el interior del exterior del objeto.

El método de fronteras (B-rep) surge a partir de los modelos poliédricos usados para la representación de objetos con eliminación de líneas y caras ocultas.

Con este esquema se representa el sólido mediante la utilización de estructuras basadas en grafos donde se almacena:

- la *geometría* de la superficie, y
- las *relaciones topológicas* entre entidades geométricas (caras, aristas y vértices).

El elemento geométrico sobre el que se basa para representar las caras en 3D es el *poliedro*: sólido limitado por polígonos donde cada arista es compartida por dos polígonos (para sólidos 2-variedad [2-manifold]).

Un punto en un objeto 2-variedad posee un conjunto arbitrario de puntos vecinos que se pueden considerar topológicamente como un disco en el plano. Existe una correspondencia uno a uno entre los vecinos del punto y el disco [Fole96].

Las estructuras de datos necesarias para representar sólidos no variedad (non-manifold) suelen ser más complejas.

2.3.3.1. Representación de B-rep

Como hemos comentado anteriormente, la representación interna utilizada en este esquema se basa en grafos, pero podemos encontrarnos con distintos tipos de estructuras:

a) Simples [Mänt88]

- *Basada en polígonos*: El sólido se representa como un conjunto de caras, y para cada cara se almacenan los vértices que la forman.

$$\text{Caras: } F_i = \{ (x_0, y_0, z_0), (x_1, y_1, z_1), \ldots \}$$

- *Basado en vértices*: Eliminación de la redundancia de los puntos que aparecen en mas de una cara. Sólo se almacenan los vértices una vez utilizando referencias a los mismos cuando sea necesario.

  Para comprobar si un punto es interior debemos ordenar (normalmente en sentido anti-horario) los vértices que forman cada cara.

  Vértices**:** $V_i = (x_i , y_i , z_i)$
  Caras**:** $F_i = (V_1 , V_2 , \dots , V_n)$

  No es necesario almacenar información asociada a la cara (p.e. ecuación de la cara). Esta se puede calcular a partir de los vértices, aunque se puede introducir para acelerar las operaciones.

  El principal problema que presenta es que es difícil conocer relaciones topológicas.

- *Representación explícita de aristas*: Almacenamos explícitamente información de las aristas para almacenar información de intersecciones sobre las caras, relaciones topológicas, etc.

  Se basa en representar el sólido como una secuencia de aristas cerradas (*loop*), que contienen punteros a los vértices.

  Caras: $F_i = \{ A_1 , A_2 , A_3 , ... \}$
  Aristas: $A_i = \{ v_1 , v_2 \}$
  Vértices: $V_i = \{ x_i , y_i , z_i \}$

b) Aristas Aladas [Mänt88]

Los métodos anteriores hacen que sean complejas ciertas operaciones por lo que, para reducir el costo computacional, se suelen utilizar estructuras más complejas entre las que podemos encontrar las *aristas aladas* (winged-edge) [Baum75].

El modelo de *aristas-aladas* se basa en las características topológicas de las aristas en un sólido 2-variedad: *cada arista es compartida por 2 caras, y en cada vértice confluyen un mínimo de 3 aristas.*

La estructura está basada en las aristas y sus conexiones topológicas con otros elementos. De esta forma, las aristas se representan por (figura 2.6):

- dos punteros a vértices,
- dos caras que comparten la arista, y
- 4 aristas que parten de ella.

Además, para cada vértice se almacena una de las aristas que parten del mismo, y para cada cara mantenemos un enlace a una de las aristas que la forman.

Fijando un orden de recorrido de los vértices, la arista se recorrerá para las dos caras, una en sentido horario y otra en anti-horario. Para cada cara se almacenan las dos aristas adyacentes, que determinan la arista anterior y siguiente.

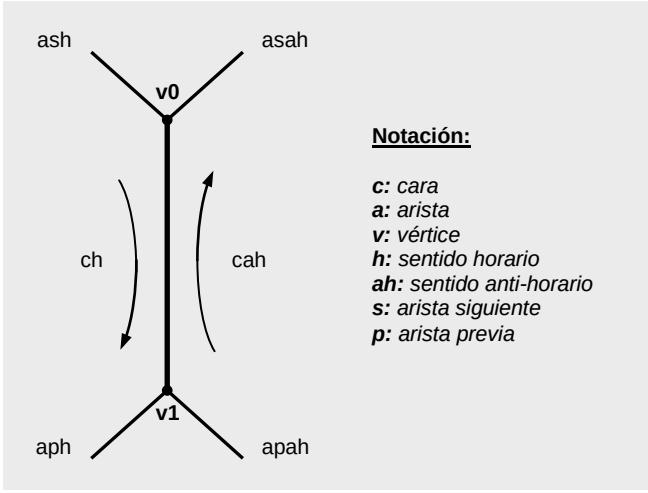


Figura 2.6. *Aristas-Aladas*: representación para cada arista.

Esta representación da acceso en tiempo constante a la relación de adyacencia entre vértices o caras, y el resto de consultas son inmediatas (p.e. aristas de una cara).

La entrada de información manual directa en este tipo de representaciones es pesada y puede ocasionar inconsistencias. Es preferible usar otro tipo de herramientas de entrada como el barrido, la instanciación o las operaciones booleanas.

La edición de modelos se suele realizar modificando los puntos de la estructura. Pero puede ocurrir que surjan problemas de validez, lo que se puede resolver, en parte, utilizando caras triangulares.

2.3.3.2. Propiedades

El *dominio* de objetos representables con este esquema depende del grado de las ecuaciones utilizadas para representar las fronteras. El caso más común es el que utiliza caras planas, que sólo permite representar sólidos poliédricos.

Es un esquema *no ambiguo* y *no único*, en el que comprobar la *validez* de las representaciones es costoso pero posible.

Una representación puede no corresponder a ningún sólido (por ejemplo si falta una cara). En el caso de caras planas, podemos comprobar la *validez* de una representación verificando las siguientes condiciones:

- Satisface la fórmula de *Euler* para los poliedros:

$$\mathbf{V - E + F = 2}$$
$$\textit{vértices – aristas + caras = 2}$$

  En caso de contener agujeros:

$$\mathbf{V + F = E + R + 2\,(B - H)}$$
$$\textit{vértices + caras = aristas + anillos + 2 (cuerpos - agujeros)}$$

  donde los *anillos* (R) son agujeros en la cara y *agujeros* (H) son pasantes. Se permiten tener más de un *cuerpo* y agujeros en un objeto.

  La fórmula de Euler es condición necesaria pero no suficiente para garantizar la validez, por lo que debemos comprobar también las siguientes restricciones.

- Restricciones geométricas:

  - Cada arista está delimitada por dos vértices.
  - Cada arista es compartida por dos caras.
  - Al menos 3 aristas se encuentran en cada vértice.
  - Las caras no se intersecan excepto en vértices comunes y aristas.

La descripción explícita de geometría y topología que permite este esquema es útil para aplicaciones de visualización, aplicación de modificaciones locales (cambiar vértice, chaflán, etc.), interrogaciones del modelo y transformaciones geométricas.

Sin embargo, las operaciones booleanas entre objetos representados por su frontera tienen una complejidad elevada debido a que, en el peor de los casos, debemos operar entre si todos los elementos geométricos.

2.3.4. Modelos constructivos: Geometría Constructiva de Sólidos

En este esquema de representación, el sólido se representa como una composición de objetos simples (primitivas genéricas, cubo, cilindro, esfera, etc.) usando operaciones booleanas y transformaciones [Requ82, Dobk88].

Es un modelo procedural en el que se especifica la forma de construcción pero no se conocen las características geométricas y topológicas del objeto final.

Para representar las primitivas se utilizan conjuntos de ***semiespacios***, definidos mediante superficies implícitas, $F(x,y,z) = 0$, donde la función F divide el espacio en dos partes, $F(x,y,z)>=0$ y $F(x,y,z)<0$.

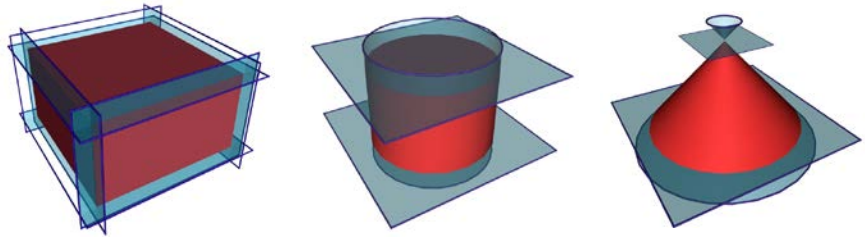


Figura 2.7. Primitivas definidas como intersección de semiespacios.

Los objetos simples se pueden representar mediante intersección de un conjunto de semiespacios (figura 2.7). Los modeladores no permiten trabajar directamente con los semiespacios, sino que ofrecen un conjunto de primitivas descritos mediante ellos.

La representación de un sólido mediante este esquema se hace usando árboles binarios ordenados (figura 2.8), donde:

- en los nodos no terminales almacenamos:
  - operaciones booleanas *regularizadas* (U^*, $\cap^*$, $-^*$), o
  - transformaciones geométricas rígidas.

- en los nodos terminales tenemos:
  - primitivas, o
  - argumentos de las transformaciones.

En este esquema no está explícitamente representada la geometría de la frontera. Esta se debe evaluar a partir de los semiespacios que definen las primitivas, proceso que es bastante costoso.

2.3.4.1. Propiedades

El *dominio* de objetos representables depende de los objetos primitivos de que dispongamos y de las transformaciones y operaciones booleanas aplicables.

Es un esquema *no ambiguo*, *no único* y *conciso.* Además, los sólidos obtenidos son *válidos* siempre que las primitivas sean válidas (conjuntos regulares y acotados) y utilicemos operaciones booleanas regularizadas (ver apartado 2.2.4).

La creación de objetos en esta estructura se hace de forma muy simple utilizando órdenes de un determinado lenguaje de descripción de árboles CSG o mediante un lenguaje gráfico que permite construir modelos complejos de una forma muy intuitiva.

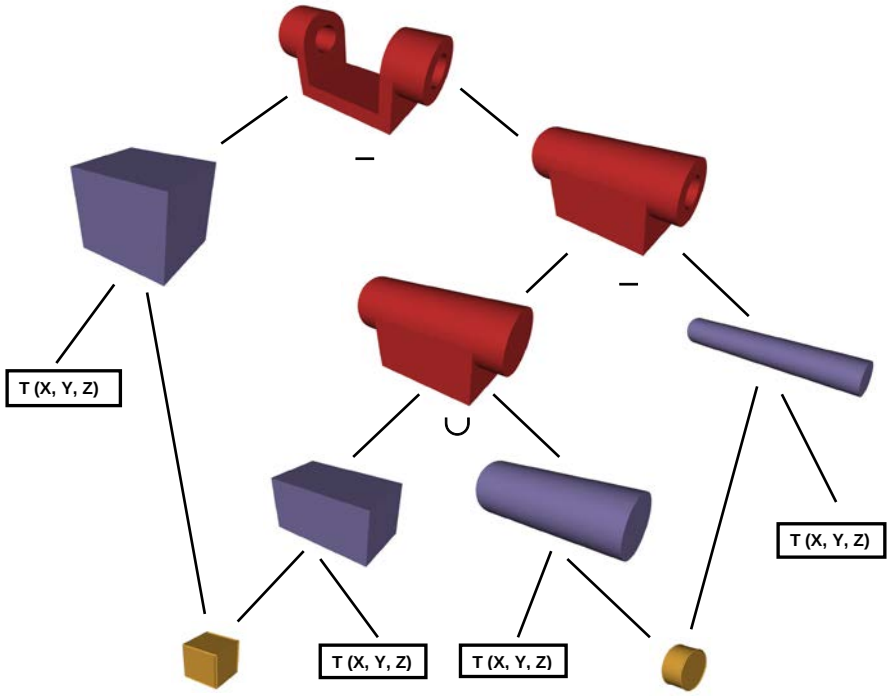


Figura 2.8. Ejemplo de modelo representado con un árbol CSG.

2.3.5. Métodos de descomposición

Son adecuados para sólidos de formas complejas y objetos volumétricos, existen una gran variedad de representaciones con características diferentes. Podemos agruparlos en dos categorías:

a) *Enumeración de retículas:* representan los objetos mediante la unión de un conjunto de celdas disjuntas y adyacentes, obtenidas a partir de la subdivisión del espacio que ocupa el propio sólido. Entre los métodos propuestos destacan:

- Descomposición celular
- Enumeración de la ocupación espacial

b) *Descomposiciones jerárquicas:* basadas en la subdivisión recursiva del espacio ocupado por el sólido, en lugar de una lista de celdas ocupadas por el sólido, se genera una estructura jerárquica que almacena el modelo. Dependiendo del tipo de celda que tenemos y de la forma en que realizamos la subdivisión tenemos, entre otros:

- BSP
- Bintrees
- Octrees
- Octrees extendidos

2.3.5.1. Descomposición celular

Representamos al sólido por la unión de las *células* en que se puede descomponer. Esas *células* son objetos más sencillos de representar que el objeto completo, basándose en la técnica *divide y vencerás* para representar modelos complejos.

Se trata de un método *no ambiguo* pero es *no único* (ver figura 2.9), y su *dominio* depende del tipo de células que podamos utilizar.

La *validez* del método, al ser una forma restringida de geometría constructiva de sólidos en la que sólo se utiliza una operación de suma sin intersección (operación de *pegado*), dependerá de la validez de las celdas en que se descomponga y del cálculo de intersecciones entre ellas.

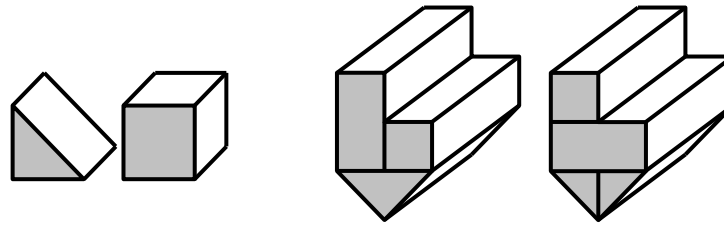


Figura 2.9. *Descomposición celular:* células usadas y no unicidad del modelo.

2.3.5.2. Enumeración

Se trata de un caso especial de la descomposición celular donde el sólido se descompone en células iguales dispuestas en una rejilla regular [Requ80]. El tipo de célula más utilizada es el cubo (*voxel*).

El espacio donde reside el objeto se divide en cubos de igual tamaño, que conforman una rejilla regular 3D. Por cada cubo (voxel), se almacena si pertenece o no al objeto [Shir81, Yau83].

Si el cubo está parcialmente ocupado por el sólido, debemos establecer un criterio de clasificación que permita decidir si tomarlo como perteneciente o no.

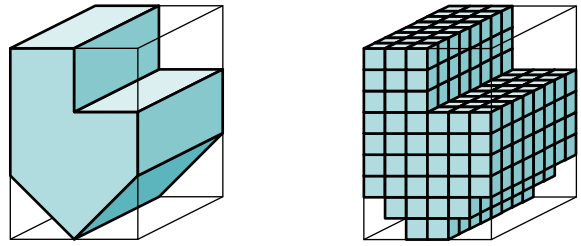


Figura 2.10. ***Enumeración de la ocupación espacial*****: sólido (izquierda) y modelo (derecha).**

La estructura de almacenamiento es muy simple: una matriz 3D de booleanos que almacena la lista de células ocupadas por el sólido (figura 2.10). Si en esa estructura no sólo almacenamos la ocupación, sino cualquier otra propiedad, el esquema se puede utilizar para representar información volumétrica.

Es un método aproximado, que presenta las ventajas de ser *único, no ambiguo,* salvo errores de aproximación, *y* fácil de *validar* e interrogar.

Para conseguir que el error cometido en la representación sea menor, debemos reducir el tamaño de las celdas, aumentando así el espacio ocupado en orden cúbico.

2.3.5.3. Bintrees

Esta representación parte de un cubo inicial (que engloba al sólido representado) y lo divide recursivamente en dos mitades iguales alternando los ejes X, Y y Z de forma cíclica como direcciones de subdivisión [Same85, Dieh88].

Se marca como *negro* la mitad que contiene sólido al completo, como *blanco* la mitad que no contiene sólido y si la ocupación es parcial, dividimos recursivamente hasta que todos los nodos queden clasificados o alcancemos un nivel máximo definido. Esto hace que la representación obtenida pueda ser *aproximada*.

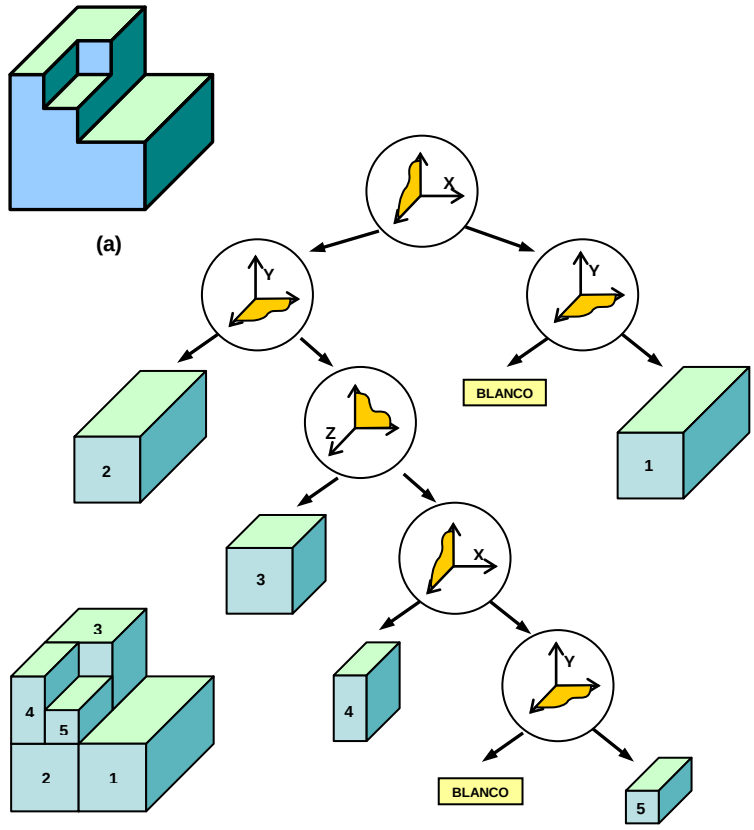


Figura 2.11. Representación de un sólido (a) mediante *Bintree*.

El modelo se representa mediante un árbol binario en el que en los nodos internos tenemos celdas parcialmente ocupadas divididas por un determinado plano y en los nodos terminales tenemos nodos totalmente ocupados o libres (figura 2.11).

Como el proceso recursivo alterna los planos utilizados en la división, la representación depende de la posición y orientación del objeto, por lo que al aplicar una transformación geométrica al modelo necesitaremos construir un nuevo árbol.

2.3.5.4. Partición Binaria del Espacio (BSP)

Inicialmente utilizado como un método para acelerar el proceso de visualización de escenas complejas [Fuch83], más adelante se ha utilizado como método de representación de objetos sólidos poliédricos [Thib87, Nayl90a, Pate90].

La *partición binaria* divide el espacio ocupado por el sólido en dos zonas mediante planos de orientación y posición arbitrarios. Es similar al *Bintree* pero en este caso los planos de división se eligen dependiendo del sólido para reducir el número de particiones [Thib87]. Los planos se eligen de forma que coinciden con los planos frontera del sólido, lo que da lugar a celdas de forma y tamaño distintas.

Se representa mediante un árbol binario (***BSP-Tree:*** *Binary Space Partitioning*) en el que en los nodos internos almacenamos la información del plano utilizado para dividir ese nodo y en los nodos terminales tendremos las celdas poliédricas interiores y exteriores al sólido.

Al utilizar planos orientados, podemos fijar que la zona interior al plano pertenezca, por ejemplo, al hijo izquierdo y la exterior al derecho (figura 2.12). Esto permite reconstruir la frontera del sólido representado de forma sencilla [Comb96].

Se trata de un esquema de representación *exacto* para objetos poliédricos, *no ambiguo* y *no único*.

Al tener celdas con forma poliédrica, los algoritmos asociados son más complejos y las operaciones booleanas se realizan mediante la inserción de planos de un árbol en otro [Thib87, Nayl90b].

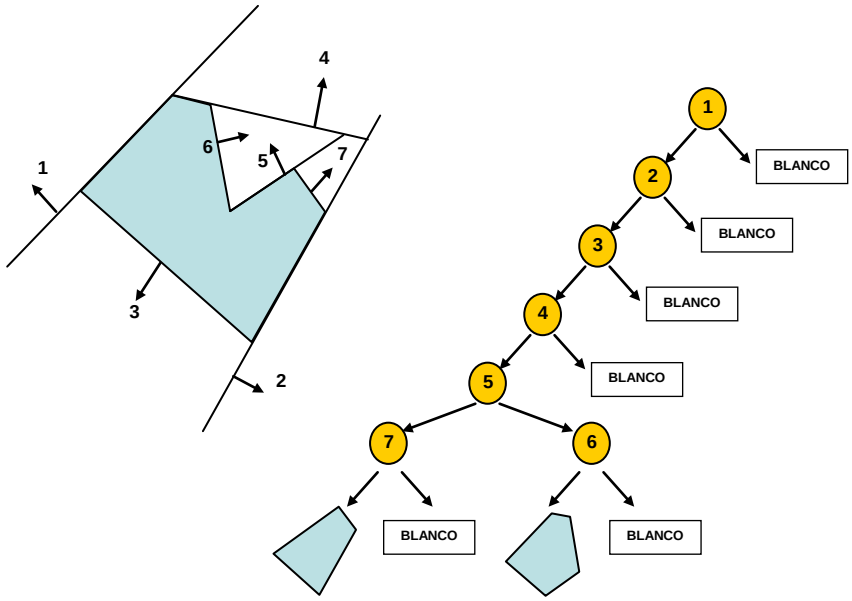


Figura 2.12. Objeto 2D representado mediante un *BSP-Tree*.

2.3.5.5. Octrees

Los Octrees son una variación jerárquica de la enumeración espacial para mejorar el problema de almacenamiento de ese esquema. Podemos verlo como una compactación de los *voxels* por coherencia espacial de forma adaptativa.

Es una generalización a 3D de los árboles cuaternarios o *Quadtrees* [Garg82b, Same84] y se basa en la división recursiva del universo cúbico que engloba al objeto a modelar.

Un Octree es la representación de un modelo mediante una estructura de árbol octal obtenida mediante divisiones recursivas de la caja envolvente del volumen a codificar en 8 cubos iguales, llamados octantes, de la mitad de tamaño. [Meag80, Meag82, Fuji84, Fuji85, Garg82a, Same88a, Same88b].

Si un octante está totalmente ocupado por el sólido, lo marcamos como ***Negro***, mientras que si está totalmente fuera se marca como ***Blanco***. Si el

octante está parcialmente ocupado se marca como ***Gris*** y se repite el proceso de división (figura 2.13).

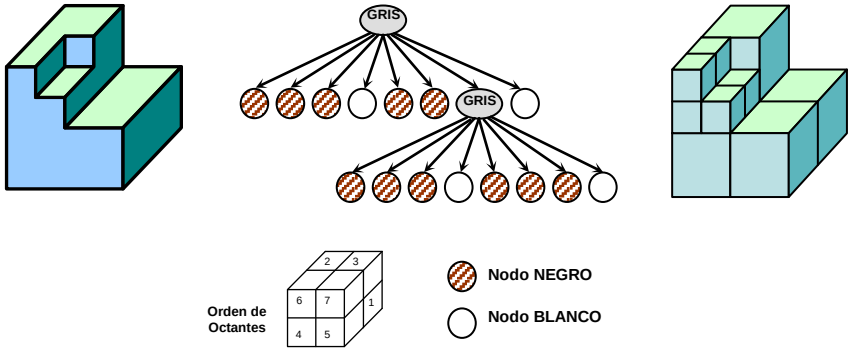


Figura 2.13. Objeto 3D representado mediante un *Octree.*

Estas divisiones se realizan de manera que los octantes obtenidos en cada nivel de la descomposición son de igual tamaño y forma, repitiendo el proceso hasta conseguir que todos los nodos del árbol sean homogéneos o hasta llegar a un nivel de profundidad preestablecido.

El tamaño mínimo de los octantes, al que llamaremos *resolución* del árbol, viene dado por el máximo nivel permitido, y sirve como condición de parada en el caso de no poder representar de forma exacta al sólido.

En el caso de que se alcance este nivel máximo, necesitamos un método para decidir la clasificación al tener ocupación parcial, lo que hace que el esquema sea aproximado.

Representación

Un octree es un árbol octal en el que cada nodo interno tiene 8 hijos. Existen varias formas de representar esa información:

a) *Como un árbol explícito* [Same88a]

Cada nodo almacena su *tipo* y 8 punteros a sus descendientes en el caso de tratarse de un nodo *Gris* (figura 2.13).

Podemos utilizar las siguientes estructuras:

```
struct raiz_octree
{
  float xmin, ymin, zmin;
  float xmax, ymax, zmax;
  struct nodo    * raíz;
}
```

```
struct nodo
{
  char código; /* N,B,G */
  struct nodo *oct[8];
}
```

donde ***raiz_****octree* almacena el nodo raíz del árbol para almacenar la caja englobante del objeto, y ***nodo*** guarda la información de cada octante. Para ahorrar espacio, para los nodos terminales podemos crear otra estructura con sólo el código (*N* o *B*).

b) *Codificación lineal* [Garg82a].

Almacenamos una lista ordenada del camino a seguir para cada nodo *negro*. Posee un dígito por cada nivel (orden de izquierda a derecha, conforme se va profundizando en el árbol) indicando en cada uno el hijo al que debemos movernos.

Para el árbol de la figura 2.13, tendríamos la siguiente lista de nodos negros donde utilizamos 2 dígitos por tener dos niveles en el árbol:

{ 1X, 2X, 3X, 5X, 6X, 71, 72, 73, 75, 76, 77 }

c) Codificación lineal en preorden [Oliv84]

Se almacena una lista ordenada de nodos generada al recorrer el árbol en preorden (primero en profundidad). Para cada nodo se indica su tipo, de forma que tras un nodo gris aparecerá la codificación de sus ocho nodos hijos.

Para el ejemplo de la figura 2.13 tendríamos la siguiente representación:

G (NNNBNNG (NNNBNNNB) B)

Dependiendo del tipo de operaciones que vayamos a realizar con el modelo será mejor utilizar una representación determinada.

Propiedades

Si bien la representación mediante Octrees es más concisa que la enumeración espacial y presenta una complejidad similar al modelo de *voxels* para las operaciones booleanas y cálculo de propiedades, se trata generalmente de una representación aproximada del sólido.

La construcción de los octrees normalmente se realiza a partir de otras representaciones [Tang88, Elbe88, Kuni85]. Las propiedades volumétricas pueden realizarse acumulando la aportación de cada nodo del árbol.

El proceso de visualización es muy simple debido al orden implícito con que almacenamos los nodos del árbol. Las infinitas posiciones del observador se pueden resumir en 8 correspondientes a los distintos octantes. Para cada una de estas clases se puede establecer un determinado orden de los nodos que permite aplicar fácilmente los métodos de visualización basados en la técnica del pintor [Doct81, Chen88, Garg89].

Los escalados del modelo son fáciles si el factor de escala es múltiplo de 2 y las rotaciones son muy sencillas si son múltiplo de 90º. En otro caso, implicaría la creación de un nuevo octree.

Las operaciones booelanas con este modelo son muy simples (al igual que en la enumeración exhaustiva) ya que no necesitan ningún cálculo geométrico. Se basan en comparar las celdas de los dos objetos operandos correspondientes a la misma región del espacio recorriendo simultáneamente los nodos de los dos árboles. El resultado depende del tipo de los nodos y de la operación a realizar.

Para el cálculo del objeto *complementario* basta con recorrer el árbol y cambiar los nodos *Blancos* por *Negros* y viceversa.

Para la intersección, al procesar dos nodos podemos encontrarnos con las siguientes posibilidades:

- Los dos nodos son terminales: si ambos son *negros*, el nodo resultante será *negro*. En otro caso, el nodo será *blanco*.
- Si uno de ellos es terminal y este es *negro*, el resultado será la copia del otro nodo. Si no, el nodo resultante será *blanco*.

- Si ninguno de los nodos es terminal, se procesan recursivamente sus nodos hijos. En este caso el resultado será un nodo *gris*, o un nodo *blanco* si al procesar los hijos el resultado en todos ellos es *blanco.*

La tabla 2.1 muestra las distintas posibilidades al realizar la intersección entre dos nodos.

∩		Sólido A		
		Blanco	Negro	Gris
Sólido B	Blanco	Blanco	Blanco	Blanco
	Negro	Blanco	Negro	*Copia* A
	Gris	Blanco	*Copia* B	*Recursión*

Tabla 2.1. Intersección entre dos nodos de modelos Octrees.

El cálculo de la unión y la diferencia puede realizarse basándonos en las dos operaciones anteriores y las propiedades de las operaciones booleanas, o utilizando tablas similares a la anterior que definan el resultado de la operación en cada caso.

En resumen, se trata de un esquema que permite representar cualquier objeto siempre que se pueda establecer un criterio de clasificación de los nodos del árbol respecto al objeto.

Sin embargo, tiene el principal inconveniente de que, salvo para casos muy concretos, la representación obtenida es aproximada.

Fijada la resolución para el árbol, el esquema es *no ambiguo*, *único*, y comprobar la validez del esquema es muy simple.

El hecho de ser aproximado hace que para obtener un error aceptable debemos utilizar un nivel de profundidad del árbol elevado, con el consiguiente incremento del espacio ocupado por el modelo.

2.3.5.6. PM-Octrees. Octrees Extendidos

Para mejorar los Octrees clásicos se han propuesto distintos esquemas jerárquicos que permiten obtener una representación exacta de un número mayor de objetos, con una menor necesidad de espacio de almacenamiento [Same90a].

Básicamente se basan en la inclusión de nuevos nodos terminales adicionales que permiten modelar de forma exacta objetos poliédricos [Ayal85, Brun85, Nava87, Brun90a]. Estos nodos almacenan información de parte de la frontera del sólido (ver figura 2.14):

- *Nodo CARA*: es un nodo atravesado por una cada plana, sin que en su interior aparezca ninguna arista ni vértice del sólido.
- *Nodo ARISTA*: almacena parte de la arista que atraviesa el nodo y la información de las dos caras que comparten esa arista.
- *Nodo VERTICE*: contiene un vértice de la superficie del objeto modelado, además de la información de las caras y aristas que convergen en él.

Dependiendo del tipo de nodos que utilice nos podemos encontrar con *Octrees Cara* (Face Octrees) [Brun90b], *Octrees Cara-Arista* (Face and Edge Octrees) [Nava89] y *Octrees extendidos*, también conocidos como *PM-Octrees*, si permite además nodos vértice [Carl85, Durs89, Ayal85].

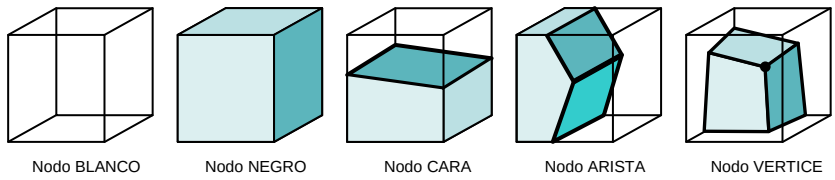


Figura 2.14. Tipos de nodos terminales en *Octrees Extendidos*.

Estos esquemas reducen el número de divisiones necesarias en el modelo, y por tanto, tienen menos requisitos de almacenamiento que los Octrees clásicos. En contra, las operaciones booleanas pierden su simplicidad al tener que realizar cálculos geométricos, aunque sigue

utilizándose la misma filosofía: se recorren los dos árboles operando los nodos homólogos de cada uno, lo que reduce el número de operaciones geométricas necesarias al tener que operar en cada caso sólo con la información de las caras que aparezcan en cada nodo [Brun90a, Ayal91, Pla93].

Además, se puede utilizar el esquema de forma generalizada para representar de forma exacta objetos con frontera no plana mediante la utilización de superficies libres (parches bicuadráticos) para la representación de las caras que forman la frontera del sólido [Brun87].

La figura 2.15 muestra la representación de dos sólidos sencillos mediante *Octrees clásicos* y *Octrees Extendidos*. En estos últimos podemos ver los nodos *vértice* en color verde, los nodos *arista* en color amarillo y los nodos *cara* en color azul.

Claramente podemos ver que el número de nodos en los árboles se ha reducido drásticamente con la utilización de los nuevos tipos de nodos, además de representar de forma exacta ambos objetos.

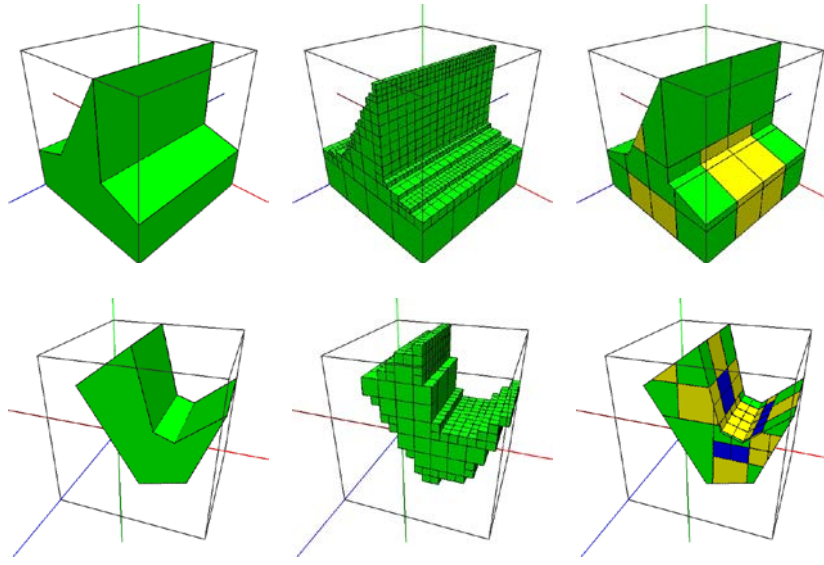


Figura 2.15. Modelos de sólidos (izquierda con *Octrees clásicos* (centro) y *Octrees Extendidos* (derecha).

2.3.6. Modelos híbridos

Cada uno de los métodos de modelado tiene aspectos positivos en los que destaca y aspectos negativos. Por ello, en la mayoría de sistemas de modelado de sólidos se utilizan diversos esquemas de representación para combinar lo mejor de cada uno de ellos en cada tipo de operación (visualización, operaciones booleanas, calculo de propiedades,...) [Mänt88].

Normalmente, en estos *esquemas híbridos*, se suele utilizar un esquema de representación *primario* y uno o varios como *secundarios*. Normalmente el usuario del sistema no tiene acceso a la representación secundaria, sino que el sistema la utiliza dependiendo de la operación que realice.

Pero este tipo de sistemas híbridos tienen dos problemas asociados: necesidad de *conversión* entre los distintos esquemas (lo que a veces puede no ser posible o tener que hacerse de forma aproximada), y *mantener la consistencia* entre las estructuras (debido a la duplicidad de información que aparece), lo que puede imponer cierta lentitud en determinadas operaciones.

2.4. Multirresolución

En la actualidad nos encontramos con la necesidad de representar objetos cada vez más complejos. Esto plantea problemas para encontrar esquemas de representación que puedan almacenar y tratar de forma eficiente ese volumen de información, así como transmitir el modelo en un tiempo razonable y utilizar distintos niveles de detalle del modelo dependiendo de la operación a realizar con el mismo.

Para resolver estos problemas en los últimos años se han diseñado diversas propuestas que podemos englobar como modelos *multirresolución* [Pupp97, Roos97a, Roos97b, Andu96, Ayal97].

Este tipo de esquemas utilizan normalmente mallas de triángulos para modelar el sólido, que pueden ser comprimidas o simplificadas mediante distintos métodos para representar con distintos niveles de detalle un mismo modelo (figura 2.16) [Hopp93, Garl99, Taub99].

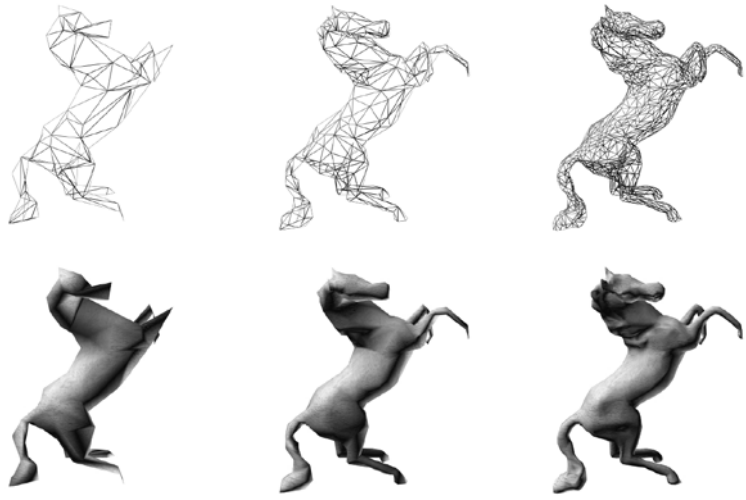


Figura 2.16. Distintos niveles de detalle en un modelo *multirresolución.*

Pero estos métodos basados en mallas multirresolución presentan un problema importante para utilizarlos como modelo de representación de sólidos. Al aplicar los algoritmos de simplificación puede ocurrir que la malla resultante no cumpla las condiciones necesarias (apartado 2.2) para que el objeto representado sea un sólido.

En la figura 2.17 vemos un ejemplo de un modelo representado con varias resoluciones (100% a la izquierda, 50% en el centro y 5% a la derecha) aplicando un método de simplificación de malla. Como vemos, en el modelo al 5% de su resolución aparecen aristas y caras aisladas, de forma que el objeto representado no cumple las condiciones de solidez.

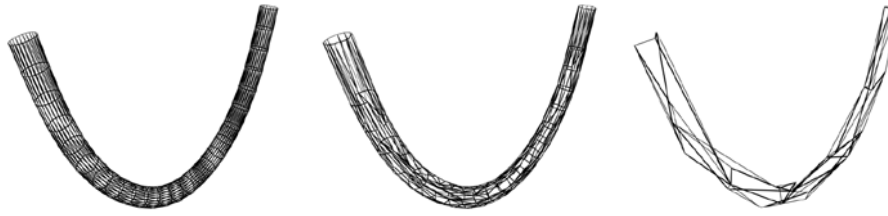


Figura 2.17. Fallo del modelo *multirresolución* para representar sólidos.

2.5. Conclusiones

En el presente capítulo se ha realizado una breve exposición del estado del arte en lo concerniente al modelado de sólidos. A partir de la descripción de las propiedades básicas de este tipo de objetos, se han descrito las características que permiten diseñar un modelo de los mismos.

Se han discutido las propiedades básicas deseables de los esquemas de representación de sólidos y se han descrito los más utilizados, centrándonos con más detalle en aquellos que han servido de base para el nuevo esquema de representación que proponemos en esta memoria.

CAPÍTULO

3

Representación jerárquica de sólidos poliédricos *SP-Octrees*

3

Desde que aparecieron los Octrees como esquema de representación de sólidos se han presentado distintas extensiones de los mismos para, manteniendo sus ventajas, mejorar los puntos débiles que presentan. En este capítulo presentamos una nueva extensión basada en la utilización de nuevos nodos terminales y la inclusión de información en los nodos internos del árbol octal. Mostraremos la estructura interna del modelo utilizado y discutiremos las características del esquema propuesto.

3.1. Introducción

Algunos de los esquemas utilizados para representar sólidos y volúmenes se basan en la descomposición del espacio, y utilizan estructuras jerárquicas para almacenar el modelo. Entre estos se encuentran la *Partición Binaria del Espacio* (BSP) y los *Octrees*.

Los árboles binarios de partición del espacio (BSP) dividen recursivamente el espacio en dos semiespacios separados por un plano con cualquier orientación y posición. Inicialmente creados para mejorar el proceso de eliminación de partes ocultas [Fuch80, Fuch83], también se han utilizado para representar objetos poliédricos de forma exacta [Thib87. Nayl90a]. Este esquema ofrece un método de representación no ambiguo, pero no único, y permite realizar operaciones booleanas de forma sencilla [Thib87, Nayl90b].

Un Octree es la representación de un modelo mediante una estructura de árbol octal obtenida mediante divisiones recursivas de la caja envolvente del objeto a representar [Meag82, Fuji84, Jack80, Fuji85, Garg82a]. Estas divisiones se realizan de manera que los octantes obtenidos en cada nivel de la descomposición son de igual tamaño y forma, repitiendo el proceso hasta

conseguir que todos los octantes obtenidos queden totalmente fuera (*nodo Blanco*) o dentro (*nodo Negro*) del sólido representado, o hasta llegar a un nivel de profundidad predefinido. Los nodos internos de la estructura se marcan como *nodos Grises*, y contienen las referencias a sus ocho octantes hijos.

La representación mediante Octrees clásicos permite realizar operaciones booleanas de forma muy sencilla recorriendo a la vez los dos árboles de los operandos y realizando la operación nodo a nodo.

Además, dado el ordenamiento espacial que podemos definir en los nodos, podemos visualizar de forma rápida el modelo, eliminando las partes que quedan ocultas, recorriendo dichos nodos en un determinado orden definido por la posición del observador respecto al objeto [Doct81, Garg86, Fole96].

Sin embargo presenta un inconveniente muy importante: las representaciones obtenidas con este esquema son aproximadas y para reducir el error cometido en esa aproximación, debemos definir un nivel de profundidad del árbol muy grande, con el consiguiente coste de almacenamiento [Same90b].

Al intentar extender el esquema de representación Octree, se pueden realizar las modificaciones del esquema en tres puntos distintos de la representación:

a) Alterar los planos de corte utilizados en el proceso de subdivisión [Cano98, Whan95].

b) Modificar la información contenida en los nodos hoja del árbol octal que representa el objeto. Para ello, podemos añadir información sobre la parte de la frontera del objeto que aparece en cada nodo terminal.

c) Modificar la información almacenada en los nodos internos del árbol.

Dentro del primer apartado nos encontrarnos trabajos en los que los octantes obtenidos en el proceso de división no son iguales debido a que los planos utilizados para realizar la subdivisión son paralelos a los ejes cartesianos, pero no dividen el universo en partes iguales al no situarlos en el centro del nodo padre [Whan95].

En este sentido, hemos estudiado también la posibilidad de modificar los Quadtrees [Same90a, Same90b] y Octrees clásicos utilizando una envolvente inicial distinta de un cubo y sistemas de coordenadas no cartesianos para realizar las divisiones recursivas (figura 3.1). Esta propuesta permite la representación

exacta de objetos con determinadas simetrías, con menor necesidad de espacio [Cano96, Torr96].

Además, se ha estudiado también la posibilidad de utilización de distintos sistemas de coordenadas al mismo tiempo, eligiendo en cada caso el más adecuado dependiendo de la geometría del objeto representado [Cano96].

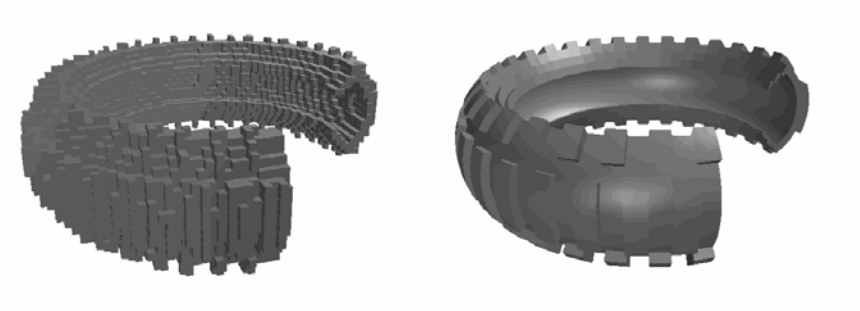


Figura 3.1. Rueda representada con Octree clásico (izquierda) y Octree con sistema de coordenadas no cartesiano (derecha).

Dentro del segundo apartado, para mejorar los Octrees clásicos se han propuesto esquemas jerárquicos que permiten obtener una representación exacta de objetos poliédricos mediante la inclusión de nuevos tipos de nodos terminales que contienen parte de la superficie del objeto, obteniendo así una representación más compacta [Brun85, Ayal85].

El último aspecto sobre el que podemos actuar en la estructura de los Octrees clásicos, es modificar la información almacenada en los nodos internos del árbol.

En los Octrees clásicos y las extensiones propuestas dichos nodos son aquellos que no son homogéneos respecto al criterio de clasificación de nodos terminales en cada esquema, por lo que en ellos sólo aparecen las referencias a los hijos resultantes en la división recursiva del nodo.

En este sentido, si almacenamos información adicional en los nodos internos del árbol, podemos utilizar dicha información para no tener que descender siempre hasta los nodos terminales.

3.2. Octrees y sólidos poliédricos

Como se comentó en el apartado anterior y podemos ver en la figura 3.2, la representación de sólidos poliédricos obtenida con Octrees clásicos es siempre aproximada, a menos que las caras frontera del objeto coincidan con las caras de los octantes en que se va dividiendo de forma recursiva el cubo inicial.

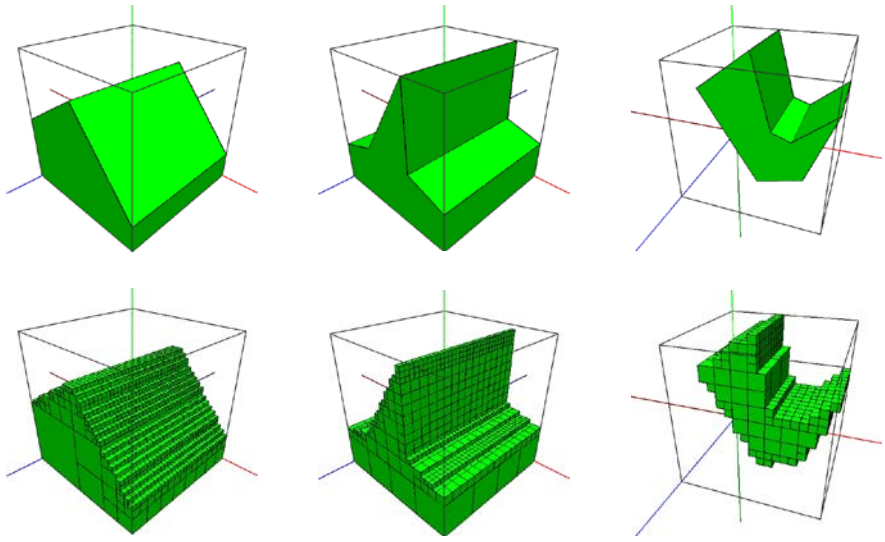


Figura 3.2. Sólidos poliédricos y su representación mediante Octrees clásicos.

Los nuevos esquemas de representación que se han desarrollado a partir de los Octrees clásicos intentan solucionar el problema de aproximación que plantean estos mediante la utilización de nuevos nodos terminales que permitan almacenar información sobre la frontera del sólido representado. De esta forma, incluyendo nodos terminales que puedan almacenar de alguna forma caras planas podemos representar sólidos poliédricos de forma exacta.

Entre estos esquemas se encuentran los *Octrees Cara*, los *Octrees Cara-Arista* y los *PM-Octrees*, dependiendo de los tipos de nodos terminales que incorpore:

- Los Octrees Cara, además de los nodos típicos de los Octrees clásicos (blancos, negros y grises), utilizan un nuevo tipo de nodo terminal, el *nodo cara*, que está atravesado por una única cara del sólido [Brun90b]. En este caso, el proceso de división se debe realizar cuando aparecen en un nodo más de una cara, aristas o vértices.
- En los Octrees Cara-Arista, además de los nodos utilizados en los Octrees Cara, aparece un nuevo nodo terminal, el *nodo arista*, que es atravesado por una arista. En ese nodo se almacena la información de la arista y de las caras que la comparten [Nav89]. La división de un nodo se realiza si aparecen más de una arista o vértices.
- Los PM-Octrees son aquellos que incorporan además el *nodo vértice*. Se trata de un nodo terminal que contiene un vértice del sólido y las caras que convergen en él.

Dentro del grupo de los PM-Octrees nos encontramos con varios esquemas: Polytrees [Carl85], Integrated Polytrees [Durs89] y Octrees Extendidos [Ayal85].

Tanto los Polytrees como los Octrees extendidos utilizan, además de los nodos Negro, Blanco y Gris, nuevos nodos terminales: cara, arista y vértice.

La diferencia primordial entre ellos es que los primeros almacenan en los nodos terminales los polígonos definidos por la intersección entre la frontera del sólido y el nodo, mientras que los Octrees extendidos no almacenan explícitamente dicha geometría. En ambos casos la representación obtenida será más concisa que la obtenida mediante Octrees clásicos.

En la figura 3.3 podemos ver los tres sólidos que aparecen en la figura 3.2 representados mediante Octrees Extendidos. En color azul vemos los planos que aparecen en nodos cara, en amarillo los planos de nodos arista y en verde los nodos vértice.

Como podemos observar, la representación obtenida es exacta y el número de divisiones necesarias es mucho menor que con los Octrees clásicos, lo que se refleja en un modelo mucho más conciso.

Sin embargo, aunque heredan la mayoría de los beneficios de los Octrees clásicos, mejoran el problema del espacio necesario para almacenar el modelo y representan de forma exacta sólidos poliédricos, los dos esquemas tienen el problema de que, cerca de aristas y vértices, es posible que se produzcan árboles

muy profundos en la representación, con el consiguiente aumento del espacio necesario para almacenar el modelo.

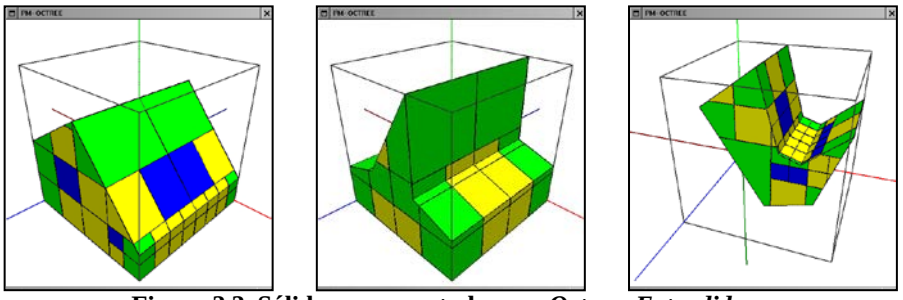


Figura 3.3. Sólidos representados con *Octrees Extendidos*.

Para evitar esto, Dursr y Kunii [Durs89] definieron los *Integrated Polytrees* que utilizan un nuevo tipo de nodo terminal con caras que convergen en una arista o en un vértice, aunque no sea en el propio nodo. Con esto se reduce considerablemente el número de nodos que aparecen en el árbol.

Además, otro inconveniente de todas las extensiones de Octrees clásicos propuestas para representar de forma exacta objetos poliédricos, es que al incluir información de la frontera del sólido sólo en los nodos terminales, los planos que aparecen dentro de un nodo gris, deberán partirse en polígonos disjuntos que se reparten en los nodos hijos del nodo al realizar el proceso de división del mismo. Por ello, la información de los planos que contienen las caras frontera del sólido normalmente aparecerá repetida en varios nodos terminales vecinos que compartan dichas caras.

3.3. Esquema de representación propuesto

La idea básica del esquema propuesto, al que hemos llamado ***SP-Octrees*** (de *Space Partition Octrees*), se basa en la modificación del esquema Octree clásico tanto en los nodos terminales utilizados como en la información almacenada en los nodos internos del árbol.

Como hemos comentado en el apartado anterior, si incluimos información de la frontera del sólido en los nodos terminales, podemos representar de forma exacta determinados objetos. Pero esa información a veces se repite en nodos terminales vecinos, como ocurre en los Octrees Extendidos.

Si tratamos de incluir también en los nodos internos información de la frontera que defina parcialmente, en ese nivel, el objeto representado en cada nodo, la información de ciertas caras frontera podrán aparecer en los niveles superiores del árbol y no será necesario repetir información en nodos terminales vecinos que comparten las caras almacenadas en sus ancestros.

Esto permitirá poder obtener información de la frontera del sólido representado en niveles superiores del árbol, lo que acelerará determinadas operaciones sobre el modelo.

3.3.1. Tipos de nodos

Utilizando la misma estructura de árbol octal que en Octrees clásicos y en Octrees Extendidos, vamos a definir un conjunto de tipos de nodos que van a permitir incluir parte de la información frontera, tanto en los nodos terminales como en los no terminales.

Básicamente, la idea es utilizar una serie de tipos de nodos distintos que almacenen planos frontera que mantienen una característica común: la concavidad o convexidad que forman entre ellos. Además, en los nodos internos se almacenará la información de los planos cuya configuración se mantiene igual en sus nodos hijos, por lo que no es necesario almacenarlos en ellos, sino que se guardan sólo en los nodos padre. Esta información permite acotar la zona del espacio en que está definido el sólido dentro del nodo.

Cuando un nodo está completamente dentro o fuera del sólido representado lo clasificamos como ***NEGRO*** o ***BLANCO***, respectivamente, de la misma forma que en Octrees clásicos (figura 3.4).

Cuando la intersección del sólido con un nodo del árbol sea convexa, utilizaremos un nodo ***CONVEXO*** (figura 3.4). En este caso, incluiremos en el nodo el conjunto de planos que definen los semiespacios cuya intersección forma el objeto convexo. Formalmente, un nodo CONVEXO es la intersección de la caja englobante del nodo con los semiespacios interiores definidos por los planos incluidos en él.

Si P_k son los semiespacios interiores de los planos definidos en el nodo CONVEXO, y *Box* es la caja englobante del nodo, la parte del sólido representada en ese nodo será:

$$R_S = \left(\bigcap_{k=0}^{n} P_k \right) \cap Box$$

Estos nodos permiten representar de forma exacta y unívoca un objeto poliédrico convexo.

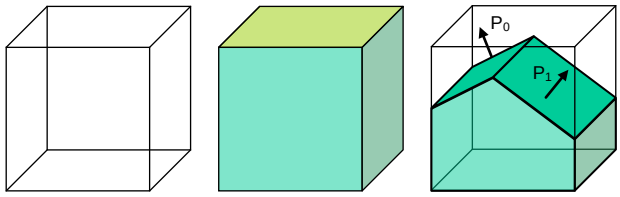


Figura 3.4. Nodos BLANCO, NEGRO y CONVEXO.

Cuando la intersección del nodo y el sólido representado sea cóncava, definiremos un nodo ***CONCAVO*** (figura 3.5). Formalmente, el sólido representado por un nodo CONCAVO será la unión de los semiespacios incluidos en él intersecados con la caja englobante.

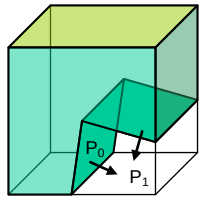


Figura 3.5. Nodo CONCAVO.

Si P_k son los semiespacios interiores definidos por los planos existentes en el nodo, y *Box* es la caja englobante del nodo, el sólido representado en un nodo CONCAVO vendrá dado por la expresión:

$$R_S = \left(\bigcup_{k=0}^{n} P_k \right) \cap Box$$

Alternativamente, podemos definir el nodo CONCAVO como la diferencia de la caja englobante del nodo con la intersección de los semiespacios incluidos en él, invirtiendo la orientación (la normal) de los semiespacios que los definen (figura 3.6).

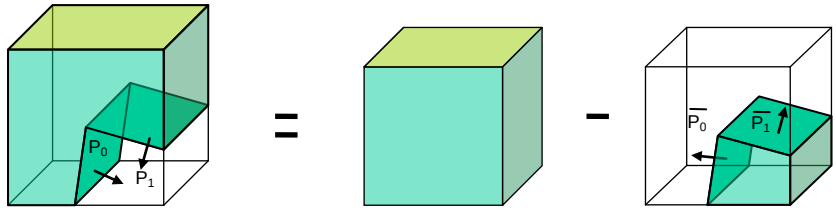


Figura 3.6. Nodo CONCAVO utilizando intersección de los semiespacios.

En este caso, podemos representar la parte del sólido que aparece dentro del nodo cóncavo mediante la expresión:

$$R_S = Box - \left(\bigcap_{k=0}^{m} \overline{P_k} \right)$$

Cuando dentro de un nodo existen concavidades y convexidades a la vez, lo clasificamos como ***GRIS***, dividiendo el nodo en ocho octantes iguales de la misma forma que en octrees clásicos, pero ahora mantenemos en el nodo la información de los planos que forman parte de la envolvente convexa de la parte de sólido representada en el nodo. De esta forma, ahora un nodo *GRIS* lo podemos ver como un nodo *CONVEXO* con hijos.

La información de los planos que mantenemos en los nodos *grises* no se repite en los distintos nodos hoja que los compartirían, sino que en ellos sólo necesitamos representar los planos frontera que no estén en esa envolvente convexa y que forman las concavidades del sólido existentes dentro del nodo (figura 3.7).

De esta forma, el sólido representado por un nodo GRIS será la unión del sólido representado en cada hijo, pero restringido a la intersección entre los

semiespacios definidos por los planos que aparecen en el nodo padre y la caja envolvente del mismo, es decir, restringidos a la parte interior al sólido representado en el nodo padre.

Siendo P_k *(k=0..n)*, los semiespacios de los planos almacenados en el nodo gris, y R_S la parte del sólido representada en el hijo ***i,*** el sólido representado por un nodo GRIS, viene dado por la expresión:

$$R_S = \left(\bigcup_{i=0}^{7} R_S^i \right) \cap \left(\bigcap_{k=0}^{n} P_k \right) \cap Box$$

En las figuras 3.7 y 3.8 vemos un ejemplo de sólido representado mediante un nodo GRIS con la ordenación utilizada para referenciar cada uno de los 8 hijos del nodo, y el árbol asociado al modelo donde podemos ver los planos incluidos en cada nodo.

Como vemos, con esta clasificación en el nodo GRIS almacenamos los planos que forman parte de la envolvente convexa del sólido intersecado con el voxel, que no están ya en ningún nodo *gris* de niveles superiores.

En un Octree Extendido deberían incluirse dentro de los nodos hijos donde aparecieran. En nuestro caso, en cada uno de los ocho hijos del nodo GRIS sólo almacenamos los planos que aparecen dentro de ellos, pero sin incluir aquellos ya almacenados en el nodo padre.

En el ejemplo de las figuras 3.7 y 3.8 vemos que los hijos 0, 1, 2, 4, 5 y 6 no contienen ningún plano, ya que los que intersecan con esos nodos ya están incluidos en el nodo padre.

Estos nodos en los que no aparece ningún plano son en realidad nodos CONVEXOS donde el sólido representado viene dado por la intersección de la caja englobante del nodo con los planos que aparecen en sus ancestros. Como al tratar un nodo debemos siempre tener en cuenta esos planos que aparecen en los nodos grises superiores, clasificamos estos nodos como NEGROS.

Los hijos 3 y 7 son nodos CONCAVOS en los que aparecen los planos P6 y P7, que son los que formaban la concavidad existente en el nodo padre.

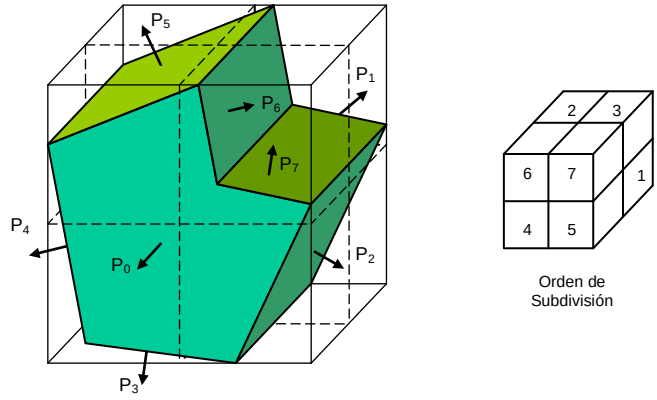


Figura 3.7. Nodo GRIS y orden establecido para los nodos hijos.

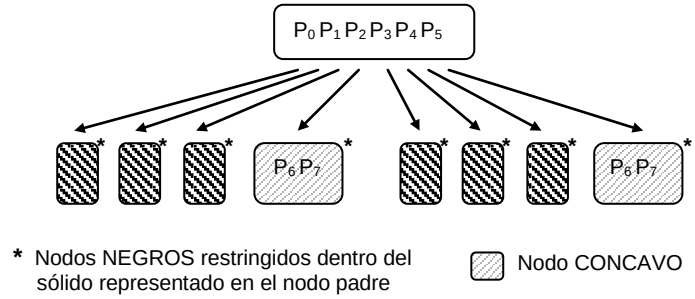


Figura 3.8. Arbol asociado al modelo de la figura 3.7.

Pero con este criterio de clasificación de los planos que aparecen en un nodo, nos podemos encontrar con situaciones en las que no sea posible clasificar el nodo como ninguno de los tipos hasta ahora definidos.

Si en el nodo existe un único vértice del sólido en el que convergen aristas cóncavas y convexas, y ninguna de las caras que comparten esas aristas pertenece a la envolvente convexa del sólido representado en el nodo, al clasificarlo como GRIS este no almacenaría ningún plano y al subdividir el nodo y repartir los planos entre los hijos, en el hijo al que perteneciera el vértice volveríamos a encontrarnos exactamente con la misma configuración.

En este caso, entraríamos en un ciclo en el que, por mucho que descendiéramos en el árbol, siempre obtendríamos un nodo gris y alcanzaríamos el máximo nivel de profundidad del árbol con una representación aproximada del sólido.

En la figura 3.9 vemos dos configuraciones típicas de un nodo con un único vértice. El nodo de la izquierda se clasifica como GRIS en el que se mantiene la información del plano que soporta las dos aristas convexas, e_{x0} y e_{x1}.

Sin embargo, el nodo de la derecha no podemos clasificarlo como GRIS porque al subdividir el nodo entraríamos en un ciclo en el que se repetiría esa misma configuración geométrica en el nodo hijo al que perteneciera el vértice, a menos que el vértice coincida con el centro del voxel.

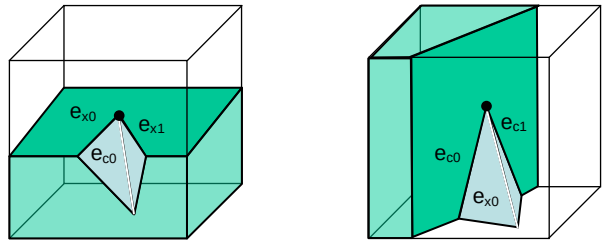


Figura 3.9. Nodos con un único vértice.

Para resolver este problema y poder representar de forma exacta cualquier sólido poliédrico necesitamos recurrir a un nuevo tipo de nodo terminal que permita representar este tipo de situaciones: el nodo ***VERTICE***.

Un nodo ***VERTICE*** será aquel que contiene un único vértice del sólido representado en el que convergen tanto aristas cóncavas como convexas, y en el que al subdividirlo con el proceso definido para los nodos grises, la misma configuración geométrica se repite en el hijo que contiene el vértice (figura 3.10).

En los nodos vértice almacenaremos, además del vértice y la información de los planos que soportan las caras que lo comparten, la configuración de cada una de las aristas que convergen en él (cóncavas o convexas). Esto permitirá clasificar un punto respecto al nodo y realizar las operaciones necesarias entre los distintos planos a la hora de visualizar u operar con el modelo [Requ85].

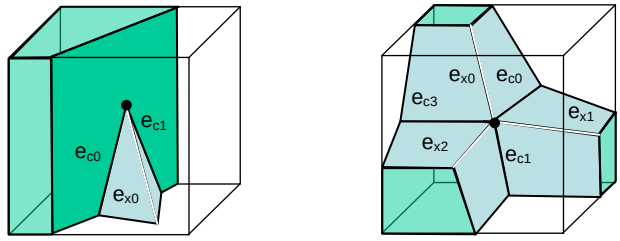


Figura 3.10. Nodos VERTICE.

Este tipo de nodo es semejante al nodo vértice utilizado en los Octrees Extendidos. No obstante, algunos nodos clasificados como vértices en estos últimos, son clasificados como grises en el nuevo modelo, almacenando parte de la información frontera en el nodo interno (figura 3.9 izquierda).

Con estos tipos de nodos mantenemos la misma expresividad que con los Octrees Extendidos ya que los nodos terminales utilizados en estos pueden ser traducidos directamente a los tipos de nodos definidos en nuestro esquema:

- Los nodos *BLANCO* y *NEGRO* son los mismos en ambos esquemas.
- El nodo *CARA* de los Octrees Extendidos es equivalente en nuestro esquema a un nodo *CONVEXO* con un único plano.
- El nodo *ARISTA* se puede traducir a un nodo *CONVEXO* con dos planos, si la arista es convexa, o a un nodo *CONCAVO* con dos planos si la arista es cóncava.
- El nodo *VERTICE* de los OE se puede expresar como un nodo *VERTICE* en nuestro esquema. Si alguna de las caras que comparten el vértice pertenece a la envolvente convexa del sólido que representa el nodo padre, esta se elimina ahora del nodo vértice al haberse incluido ya en dicho nodo padre.

  Si algunas de las caras que comparten el vértice pertenecen a la envolvente convexa, entonces sería equivalente a un nodo *GRIS* del esquema propuesto (figura 3.9 izquierda).
- El nodo *CASI-VERTICE*, definido para mejorar los Octrees Extendidos originales evitando la aparición de nodos grises de máximo nivel [Brun90a], tiene una equivalencia directa en nuestro esquema a través

del nodo *CONVEXO*, si no hay concavidades, o un nodo *GRIS* con hijos *CONCAVOS* si las hay.

La tabla 3.1 muestra un resumen de las posibles equivalencias entre los nodos terminales de un Octree Extendido y los de un SP-Octree.

Nodo Octree Extendido	Nodo SP-Octree
BLANCO	*BLANCO*
NEGRO	*NEGRO*
CARA	*CONVEXO*
ARISTA	*CONVEXO* si arista convexa *CONCAVO* si arista cóncava
VERTICE	*CONVEXO* si sólo hay aristas convexas *CONCAVO* si sólo aparecen aristas cóncavas *VERTICE* si aristas cóncavas y convexas *GRIS* si alguna cara pertenece a la envolvente convexa
CASI-VERTICE	*CONVEXO* si no hay concavidades *GRIS* si hay concavidades

Tabla 3.1. Equivalencia de tipos de nodos entre Octrees Extendidos y SP-Octrees.

Esta equivalencia propuesta nos permitirá de forma sencilla convertir modelos representados mediante Octrees Extendidos al esquema SP-Octree y viceversa, como veremos en el siguiente capítulo.

3.4. Representación interna

Para representar internamente el esquema propuesto utilizamos las siguientes estructuras de datos, de forma similar a como se utilizan en los Octrees Extendidos [Ayal85, Brun85]:

- Una matriz con las ecuaciones de los planos, $ax+by+cz+d=0$, que definen los semiespacios utilizados en cada nodo para representar el sólido.

  La normal a dichos planos se utilizará para distinguir el interior y el exterior de cada semiespacio. Definimos el *interior* de un semiespacio

como el conjunto de puntos *(x,y,z)* que satisfacen la desigualdad $ax+by+cz+d<0$.

- La codificación del árbol octal donde:

  a) En los nodos terminales, almacenamos el tipo de nodo, el número de planos incluidos en él y una referencia a cada uno de ellos en la matriz auxiliar de planos.

     En el caso del nodo vértice almacenaremos también las coordenadas del vértice y la configuración de las aristas que convergen en él.

  b) En los nodos internos almacenamos el número de planos que están en ese nodo y la referencia a la posición de los mismos en la matriz auxiliar. Además, almacenaremos un enlace a los ocho hijos.

     Los nodos hijos pueden contener la información de los planos que forman concavidades en el nodo padre, y podrán ser tanto nodos terminales como internos.

Como elementos de la matriz de planos tenemos vectores de cuatro flotantes que almacenan los coeficientes de cada plano. Esa matriz se puede ir creando a la vez que construimos el modelo o completarla desde el principio almacenando todos los planos frontera del sólido a representar.

Además, esta matriz permite evitar duplicados, de forma que si varias caras frontera del sólido pertenecen al mismo plano, sólo almacenamos su ecuación una única vez.

La codificación del árbol podemos realizarla con cualquiera de los métodos descritos en el capítulo 2 para Octrees clásicos. En nuestro caso hemos elegido utilizar una representación explícita del árbol pero sin utilizar la estructura de datos *puntero* para referenciar a padres e hijos.

Para ello utilizamos dos matrices, una para los nodos internos y otra para los terminales, donde las referencias entre nodos se realizan mediante aritmética entera, indicando en cada caso la posición en esas matrices del nodo referenciado (figura 3.11).

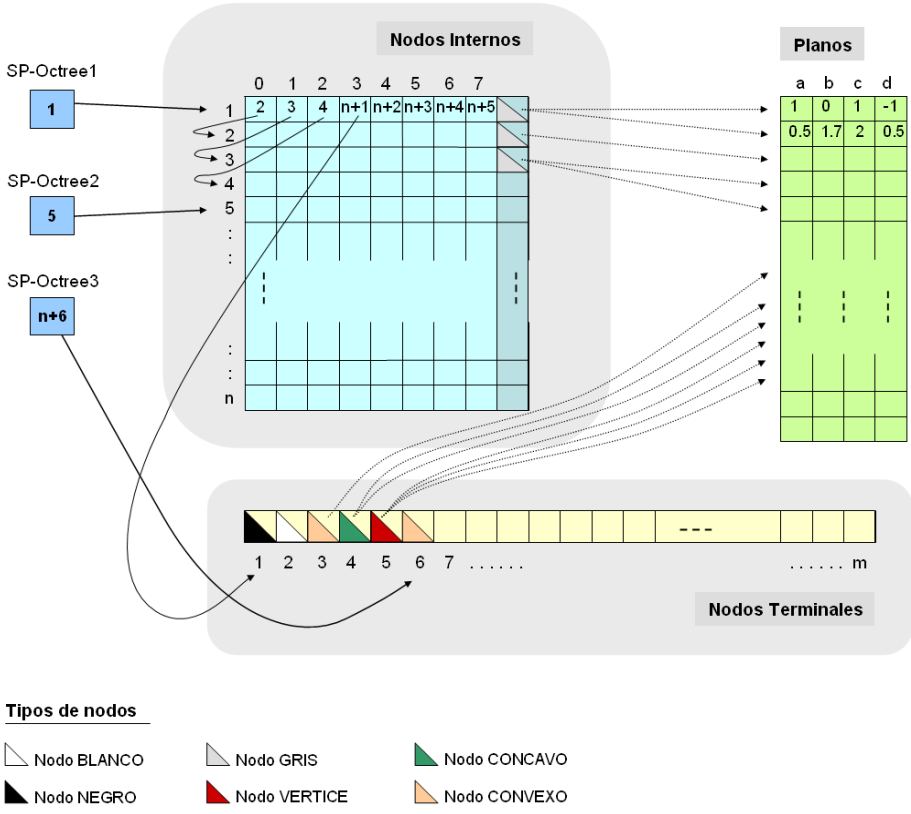


Figura 3.11. Estructura interna para almacenar el modelo.

En esta estructura definimos un número máximo de nodos internos y de nodos terminales, de forma que si el valor que aparece como referencia a un nodo es mayor que el número máximo de nodos internos, el nodo referenciado será terminal. Estos valores máximos definen el número *k* de bits necesarios para codificar las distintas posiciones de las matrices.

Para reducir el espacio de almacenamiento utilizado, estas matrices pueden crearse de forma dinámica, aunque el valor límite de nodos internos debe definirse inicialmente para poder utilizar la referencia descrita a los nodos terminales.

En cada nodo interno almacenamos las referencias a sus 8 nodos hijos, junto con el número de planos que aparecen en el nodo y la lista de referencias que indican la posición de esos planos en la matriz de planos auxiliar. El número de bits, *j*, necesarios para codificar esas referencias dependerá del número máximo de planos que podamos almacenar en la matriz auxiliar.

En los elementos de la matriz de terminales almacenamos el tipo de nodo, el número de planos pertenecientes al nodo y una lista de referencias a los planos en la matriz.

Los nodos cóncavos almacenarán la referencia a los planos que forman concavidades en el sólido, y será en el momento de tener que procesarlos cuando tendremos que invertir sus normales para poder tratarlos según la definición que hemos hecho en el apartado anterior.

En los nodos vértices, debemos utilizar además *n* bits (siendo *n* el número de caras que comparten el vértice) para codificar la configuración, cóncava o convexa, de cada una de las aristas que convergen en el vértice, considerando siempre un ordenamiento cíclico de las mismas para poder después operar con los planos asociados de forma cómoda.

Como los nodos internos se almacenan en una estructura distinta a los nodos terminales, no es necesario codificar su tipo. En la matriz de terminales, al tener 5 tipos de nodos distintos (Blanco, Negro, Cóncavo, Convexo y Vértice), necesitamos 3 bits para codificarlos.

Sin embargo, podríamos fácilmente utilizar sólo 2 bits codificando los nodos convexos y cóncavos con el mismo código (tratándolos como si fueran un único tipo de nodo), y añadiendo, sólo en esos nodos, un nuevo bit que definiera la configuración de los planos que aparecen en ellos: 0 para cóncavos y 1 para convexos. Igualmente podríamos hacer si codificamos los nodos convexos y negros como un único tipo de nodo, ya que como vimos en el apartado 3.3 los nodos negros podemos tratarlos como nodos convexos sin planos que estarán recortados por los semiespacios que aparecen en sus ancestros.

La tabla 3.2 muestra un resumen del número de bits necesarios para almacenar cada uno de los tipos de nodos utilizados en el esquema, siendo *k* el número de bits necesarios para codificar las referencias a nodos hijos, *j* el número de bits necesarios para codificar la referencia a los planos y *n* el número de planos que aparecen en un nodo.

Tipo de nodo	Bits necesarios
BLANCO	3
NEGRO	3
CONVEXO	3 + (n+1)*j
CONCAVO	3 + (n+1)*j
VERTICE	3 + (n+1)*j + n
GRIS	(n+1)*j + 8*k

Tabla 3.2. Tamaño necesario para almacenar cada tipo de nodo.

Al almacenar en cada nodo sólo las referencias a los planos de la matriz auxiliar, y al no almacenar internamente la geometría del objeto representado sino sólo las ecuaciones de los planos frontera, la representación obtenida es compacta y reduce las necesidades de almacenamiento

Aunque en la figura 3.11 no aparecen, para determinadas operaciones se hace necesario acceder a la información de los ancestros de un nodo, por lo que si queremos facilitar ese acceso podríamos añadir a cada nodo una referencia a su nodo padre, lo que supondría un aumento del espacio ocupado a cambio de ganar en velocidad al recorrer la estructura.

Esta estructura nos permite además almacenar varios modelos a la vez, de forma que si comparten planos frontera no es necesario repetirlos, sino que aparecen una única vez en la matriz auxiliar de planos.

En la figura 3.11 aparecen tres modelos distintos, definidos por la referencia a su nodo raíz en la estructura. Los dos primeros serían modelos cuya raíz es un nodo gris, mientras que el tercero es un sólido representado por un único nodo terminal convexo.

3.5. Construcción del modelo

El algoritmo de construcción de un modelo en el esquema propuesto es muy similar al de construcción de un Octree Extendido, con diferencias en dos puntos importantes: el tratamiento de los nodos grises y el proceso de clasificación de un nodo (por la existencia de nuevos tipos de nodos terminales).

En los Octrees Extendidos la clasificación de los nodos se realiza en función del número de caras, aristas y vértices que aparecen en el nodo. En nuestro caso, además se utiliza la configuración de todas las aristas que aparecen en el nodo.

A partir del sólido que deseamos representar con el esquema propuesto se calcula su caja englobante y los coeficientes de los planos que soportan todas las caras frontera del sólido.

Esta información geométrica de los planos se almacena en la estructura auxiliar utilizada para no duplicar información al encontrarnos con nodos que compartan planos. Como podemos tener a la vez distintos modelos que comparten la misma estructura auxiliar, cada vez que incluimos un nuevo plano en esa estructura se debe comprobar que no esté ya incluido para evitar información duplicada.

Con esa información como entrada se pasa a construir la estructura jerárquica que representa el modelo según el siguiente algoritmo recursivo. Partiendo de la caja que engloba el sólido a representar, que es la caja asociada al nodo raíz del árbol, de las caras frontera del objeto incluidas en el nodo y de la lista de vértices del sólido pertenecientes al mismo, se clasifica dicho nodo atendiendo a las siguientes situaciones:

1. *Nodos terminales:*

   Si el nodo está completamente fuera o dentro del sólido se clasifica como BLANCO o NEGRO como en Octrees clásicos. Para ello, necesitamos calcular la pertenencia del nodo al sólido, lo que podemos obtener fácilmente a partir de la configuración de los nodos vecinos.

   En este caso se crea un nuevo nodo terminal en la estructura jerárquica almacenando sólo el código del tipo de nodo.

   Si todas las aristas que aparecen en el nodo son convexas/cóncavas, el nodo se clasifica como CONVEXO/CONCAVO.

   En estos dos casos se crea un nuevo nodo terminal en la estructura, almacenando el código del tipo de nodo y las referencias a las posiciones que ocupan en la estructura auxiliar los planos que aparecen en dicho nodo.

Si existen aristas cóncavas y convexas a la vez, pero todas comparten un vértice del sólido, y éste es el único que pertenece al nodo, lo clasificamos como VERTICE.

En este caso se crea un nuevo nodo terminal que almacena el código del tipo de nodo VERTICE, las coordenadas del vértice y las referencias a la posición que ocupan en la estructura auxiliar de los planos que comparten ese vértice.

2. *Nodos internos:*

Cuando en el nodo aparecen aristas cóncavas y convexas, y más de un vértice del sólido, se creará un nodo *GRIS*, subdividiendo el cubo en ocho octantes iguales de la misma forma que en Octrees Clásicos y Extendidos.

Si sólo aparece una arista cóncava y varias convexas, o las aristas cóncavas son todas vecinas, es decir, no hay entre ellas aristas convexas, existirán una serie de caras que pertenecen a la envolvente convexa. En este caso no se creará un nodo vértice, sino que también se clasificará el nodo como GRIS.

En la figura 3.12 vemos las dos situaciones posibles cuando aparece un único vértice en un nodo. En el caso inferior hay más de una arista cóncava alternándose con aristas convexas, y ninguna de las caras pertenece a la envolvente convexa del nodo, por lo que se creará un nodo vértice. En el caso superior, nos encontramos con un nodo cóncavo (con dos planos –en color azul- que comparten la única arista cóncava que converge en el vértice común) y con tres planos (en color amarillo) incluidos en el nodo gris padre de este nodo cóncavo.

En ese nodo almacenamos la referencia a sus ocho nodos hijos, y la referencia a los planos que pertenecen a la envolvente convexa de la parte del sólido representado en el nodo. Estos planos se obtendrán de la lista de caras de entrada al nodo.

Al igual que en Octrees Extendidos, para procesar recursivamente los ocho nodos hijos, debemos repartir entre ellos las caras y vértices que aparecen en el nodo padre, eliminando aquellas caras que ya se han tenido en cuenta en el nodo GRIS.

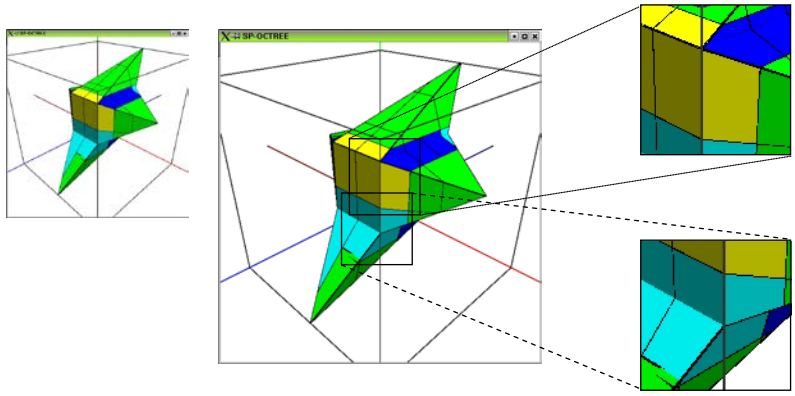


Figura 3.12. Nodos *Gris* (superior) y *Vértice* (inferior) con un vértice.

Una vez eliminadas las caras referenciadas en el nodo padre, y dividida la información restante entre los nodos hijo, se calcula la caja englobante de cada uno de los nodos y se procesan recursivamente cada uno de ellos con el mismo algoritmo descrito.

Si se ha impuesto un máximo nivel permitido para el árbol, y este es alcanzado, no se podrán crear nuevos nodos hijos, almacenando en el nodo GRIS sólo la referencia a las caras que forman la envolvente del sólido representado en el nodo. En este caso, crearemos un nuevo nodo terminal CONVEXO, obteniendo una representación aproximado del sólido.

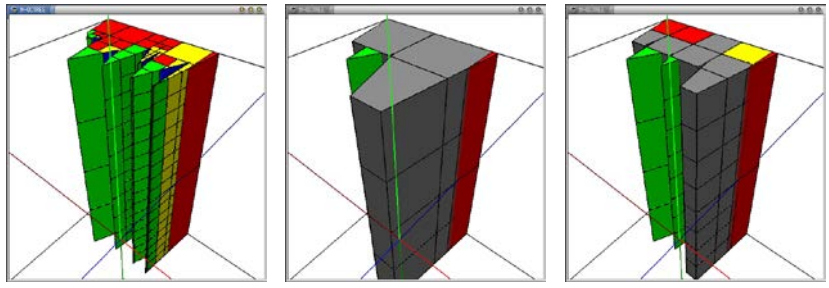


Figura 3.13. SP-Octree con profundidad 6 (izquierda) y aproximación con 3 (centro) y 4 (derecha) niveles.

En la figura 3.13 vemos un ejemplo de un SP-Octree de 6 niveles y la aproximación obtenida si limitamos la profundidad máxima a 3 y 4 niveles. En color gris se representan los nodos grises de último nivel clasificados como convexos.

En el algoritmo 3.1 podemos ver el pseudocódigo de este proceso de construcción. A continuación describiremos con más detalle los aspectos más importantes del mismo.

- Esta función recibe como parámetros de entrada la caja englobante del nodo, la lista de caras y vértices del sólido pertenecientes a ese nodo, y el nivel de profundidad del nodo a construir (inicialmente el máximo permitido para el árbol).

  Como salida devuelve una referencia a un nuevo nodo de la estructura, que puede ser tanto interno como terminal. Es fácil comprobar que el nodo raíz del árbol será siempre un nodo interno gris o un nodo terminal convexo.

- La función ***NuevoTerminal*** crea un nuevo elemento en la lista de nodos terminales, almacenando el tipo de nodo y, según ese tipo, la referencia a los planos que soportan las caras que aparecen en el nodo y las coordenadas del vértice asociado al nodo (en el caso de crear un nodo vértice).

- La función ***Configuración*** calcula la concavidad o convexidad de las aristas que forman la lista de caras de entrada. Para ello calculamos el ángulo que forma cada par de caras que comparten una arista.

- Si todas las aristas son convexas, todas las caras pertenecen a la envolvente convexa de la parte del sólido representado en el nodo, por lo que se crea un nodo terminal *convexo*. Si todas las aristas son cóncavas, se creará un nuevo nodo terminal *cóncavo*.

- Si tenemos tanto aristas cóncavas como convexas, debemos estudiar si el nodo es un nodo *vértice* (cuando sólo existe un vértice dentro de él y además todas las caras que aparecen en el nodo lo comparten) o es un nodo *gris.*

```
FUNCTION ConstruyeSPO(Box, ListaCaras, ListaVertices, MaxLevel, SPO)
{
  if (ListaCaras==EMPTY)
    if (Box es interior al Sólido)
      SPO=NuevoTerminal(NEGRO);
    else
      SPO=NuevoTerminal(BLANCO);
  else
    if (Configuración(ListaCaras)==Convexas)
      SPO=NuevoTerminal(CONVEXO, ListaCaras);
    else
      if (Configuración(ListaCaras)== Concavas)
        SPO=NuevoTerminal(CONCAVO, ListaCaras);
      else
      {
        if ((NumVertices(ListaVertices ∈ Box)==1) &&
            (Todas las caras de ListaCaras comparten (Vértice ∈ Box))
        {
          SPO=NuevoTerminal(VERTICE, ListaCaras, Vértice);
        }
        else
        {
          QuitarConcavidad (Box, ListaCaras, LConvexas, LConcavas);

          if (MaxLevel==0)
            SPO=NuevoTerminal(CONVEXO, LConvexas);
          else
          {
            SPO=NuevoSPO (LConvexas);

            Divide (LConcavas, Caras[i], ListaVertices, Vert[i]);

            for i=0 to 8 do
            {
              Box[i]=Determinar caja englobante del nodo hijo i;
              ConstruyeSPO (Box[i], Caras[i], Vert[i], MaxLevel-1,
                            SPOhijo[i]);
              AsignaHijo(SPO, i, SPOhijo[i]);
            }
          }
        }
      }
}
```

Algoritmo 3.1. Construcción de un SP-Octree.

- En el caso de tener un nodo gris, comprobamos primero si hemos alcanzado el máximo nivel de profundidad del árbol, en cuyo caso creamos un nuevo nodo terminal convexo con los planos pertenecientes a la envolvente convexa del sólido representado en el nodo. Como hemos comentado antes, el resultado en este caso será una aproximación del sólido.

```
FUNCTION QuitarConcavidad (Box, Caras, ListaConvexas,
                                       ListaConcavas)
{
  for i=0 to NumCaras(Caras) do
  {
    int a_convexas=a_concavas=0;

    for cada arista de Caras[i]
    {
      if (arista ∈ Box)
        if (arista==convexa)
          a_convexas++;
        else
          a_concavas++;
    }

    if (a_concavas==0)
    {
      cara_convexa=TRUE;

      for j=0 to NumCaras(Caras) do
        if (i!=j)
          if (Plano(Cara[i]) interseca Cara[j])
            cara_convexa=FALSE;

      if (cara_convexa=TRUE)
        Insertar Caras[i] en ListaConvexas;
   else
     Insertar Caras[i] en ListaConcavas;
    }
    else
      Insertar Caras[i] en ListaConcavas;
  }
}
```

Algoritmo 3.2. Cálculo de envolvente convexa en un nodo.

- La función ***QuitarConcavidad*** devuelve, a partir de un conjunto de caras, dos nuevas listas: una con aquellas caras que forman parte de la envolvente convexa del sólido representado en el nodo, y otra con aquellas que comparten alguna arista cóncava o que, aunque no compartan aristas cóncavas, su plano soporte corta alguna otra cara, por lo que no pertenece a la envolvente convexa (ver algoritmo 3.2).

- Una vez obtenidas las caras que forman parte de la envolvente convexa de toda la parte del sólido representada en el nodo, se crea un nuevo nodo interno mediante la función ***NuevoSPO***.

  Esta función inicializa un nuevo elemento libre de la estructura de nodos internos, almacenando la referencia a los planos que contienen las caras que forman parte de la envolvente convexa.

- Con la función ***Divide*** repartimos las caras y vértices de un nodo padre entre sus hijos. Las caras que debemos repartir entre los hijos son aquellas que no se han incluido ya en el nodo padre.

  Para ello se calcula si dichas caras intersecan, están totalmente dentro o fuera de cada nodo hijo. Como alguna cara puede intersecar más de un nodo, puede ocurrir que una misma cara aparezca en más de un hijo.

  Para evitar redundancias entre nodos vecinos al realizar la clasificación, las fronteras izquierda, trasera e inferior del cubo envolvente del nodo se consideran como pertenecientes al nodo, mientras que las fronteras derecha, delantera y superior se consideran no pertenecientes, ya que se tomarán como pertenecientes al nodo vecino.

- La llamada recursiva a la función ***ConstruyeSPO*** se realiza con la caja englobante de cada nodo hijo, su lista de caras y vértices, y decrementando en 1 el valor del nivel de construcción, ya que hemos descendido un nivel en la estructura.

- Por último, la función ***AsignaHijo*** enlaza el nodo gris padre con cada nodo hijo construido.

3.6. Algunos ejemplos

A continuación vamos a presentar algunos ejemplos de modelos creados en el esquema propuesto para facilitar la comprensión del algoritmo de construcción presentado y de la representación utilizada.

En todos los ejemplos aparece representada la caja englobante inicial, y hemos utilizado la misma codificación de color para representar los polígonos asociados a planos de los distintos tipos de nodos:

- Los planos pertenecientes a nodos *convexos* se visualizan en color ***verde***. Los planos pertenecientes a los ancestros de estos nodos también se representan en verde.
- Los planos pertenecientes a nodos *cóncavos* se representan en color ***azul***. En este caso, los planos que aparecen en los nodos ancestros de este tipo de nodos se visualizan en ***amarillo.***
- Los planos de nodos *vértice* aparecen representados en color ***celeste.*** De nuevo, utilizamos el color ***amarillo*** para los planos que aparecen en sus nodos ancestros.
- Para los nodos negros utilizamos el color ***Rojo*** para representar los planos existentes en sus ancestros.
- Por último, si con algún nodo *gris* hemos alcanzado el nivel de profundidad máximo establecido para el modelo, aunque estos se crean como nodos terminales *convexos*, para distinguir la aproximación que generan se ha optado por utilizar otra codificación en la visualización.

  En este caso utilizamos el color ***morado*** para representar los planos que aparecen en esos nodos, y el color ***gris*** para los planos frontera del voxel utilizados para cerrar los agujeros que se generan al intersecar entre si los planos incluidos en el nodo.

Para mejorar la visualización del modelo no se representan las aristas de los cubos frontera de cada nodo, ni los nodos negros y blancos (totalmente dentro y fuera del sólido).

En la figura 3.13 podemos ver tres objetos poliédricos representados por SP-Octrees de un nivel (izquierda), dos niveles (centro) y tres niveles (derecha). Como podemos observar, el árbol del ejemplo de la izquierda está formado por un único nodo terminal en el que almacenamos la referencia a los planos soporte de todas las caras frontera del sólido.

En el ejemplo del centro, los cinco planos delantero, trasero, izquierda, derecha e inferior del sólido se almacenan en el nodo raíz, por lo que en los hijos sólo se almacenan los dos planos que comparten la arista cóncava y el plano superior que, aunque no comparte ninguna arista cóncava, no pertenece a la envolvente del sólido en el nodo raíz ya que su plano soporte interseca con otra cara del sólido en ese nodo.

El modelo de la derecha de la figura 3.14 presenta un vértice en el que convergen aristas convexas y más de una arista cóncava. En ese caso se observa como el nodo superior derecho se clasifica como vértice. En él sólo aparecen tres planos, ya que los otros tres que intersecan el nodo ya están incluidos en el nodo gris padre (polígonos en color amarillo).

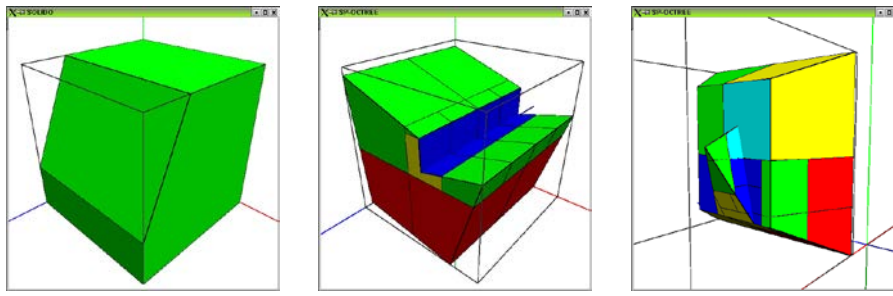


Figura 3.14. SP-Octrees de 1 (izquierda), 3 (centro) y 4 (derecha) niveles.

En la figura 3.15 se muestran otros tres ejemplos de sólidos en los que aparecen nodos *vértice*. Como podemos observar, en el modelo de la izquierda los nodos vértice no presentan ningún plano en sus ancestros. Sin embargo, en el modelo central y en el de la derecha, el nodo vértice que aparece en ambos casos contiene varios planos formando distintas zonas cóncavas disjuntas. En estos dos modelos el nodo gris padre del nodo vértice contiene los planos que engloban el sólido representado en dicho nodo.

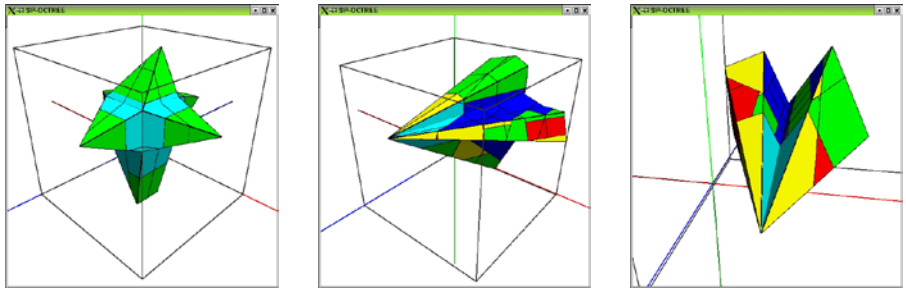


Figura 3.15. SP-Octrees con nodos *vértice*.

El esquema de representación presentado permite también representar objetos con más de un cuerpo. La figura 3.16 muestra tres ejemplos de estos modelos.

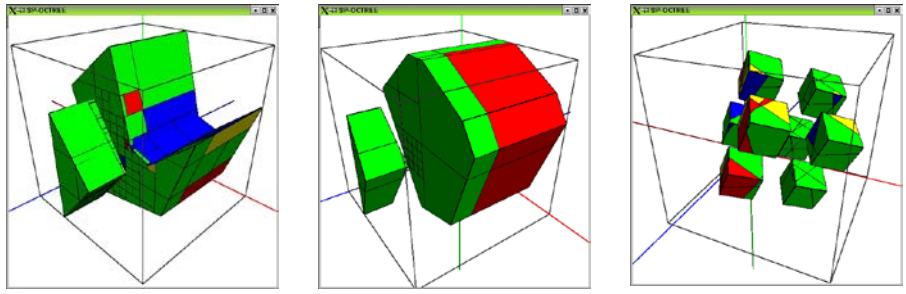


Figura 3.16. SP-Octrees de sólidos con varios *cuerpos*.

Los dos primeros están formados por dos cuerpos y en el modelo, como vemos en las imágenes, el mayor nivel de subdivisión se presenta en las zonas donde la distancia entre los dos cuerpos es menor. Esto se debe a que esos nodos, hasta que no separemos las caras de cada cuerpo, el proceso de construcción definido los clasifica como *grises*, por lo que es necesario subdividirlos hasta un tamaño que permita tratar los planos de cada cuerpo en nodos distintos.

Con la definición realizada de los distintos tipos de nodos terminales, el esquema permite también la representación de forma exacta de objetos poliédricos con agujeros.

En la figura 3.17 vemos algunos ejemplos de sólidos con agujeros representados mediante SP-Octrees, donde podemos observar como los planos interiores a los agujeros pertenecen, en estos casos, a nodos *cóncavos* cuyos ancestros contienen los planos exteriores de los agujeros.

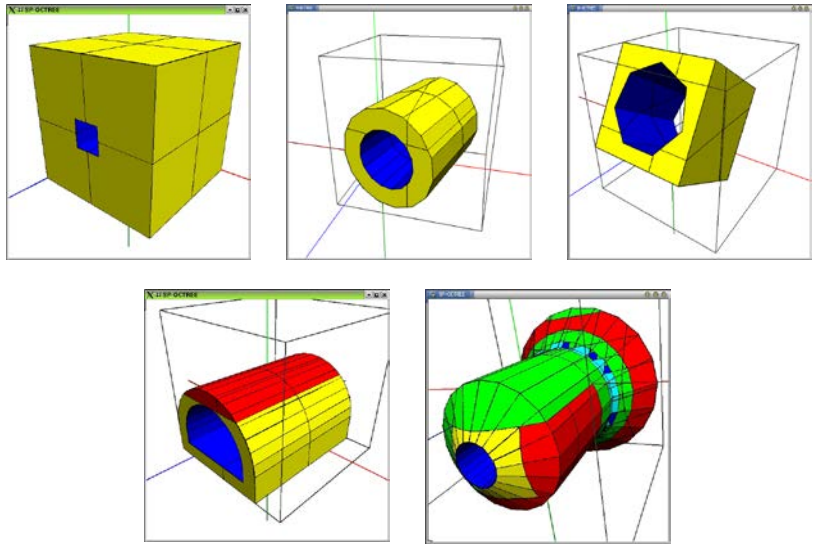


Figura 3.17. SP-Octrees de sólidos con agujeros.

Por último, las figuras 3.18 y 3.19 muestran la representación de una pieza mecánica y una escultura de un conejo con dos niveles distintos de profundidad del SP-Octree utilizado.

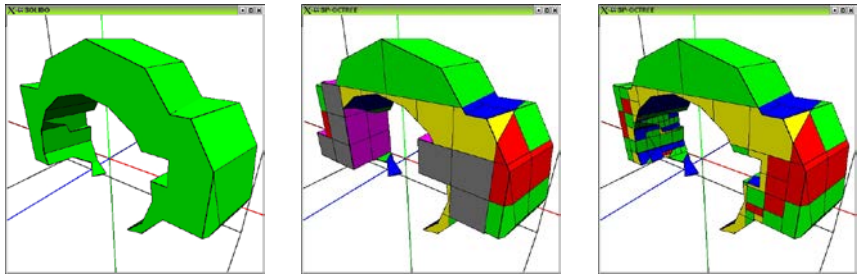


Figura 3.18. *Pieza mecánica*: SP-Octrees de 3 (centro) y 6 (derecha) niveles.

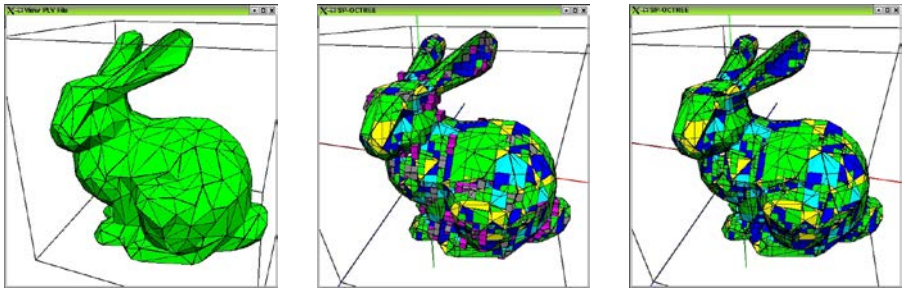


Figura 3.19. *Conejo*: SP-Octrees de 4 (centro) y 11 (derecha) niveles.

En las imágenes del centro tenemos modelos aproximados de los sólidos debido a que se ha construido un árbol de 3 y 4 niveles de profundidad respectivamente.

Como vemos, en los nodos *grises* de último nivel, que han sido clasificados como *convexos*, la representación obtenida de la parte del sólido que pertenece a esos nodos es sólo una aproximación definida por los planos que pertenecen a la envolvente convexa de esa porción de sólido.

En las imágenes de la derecha mostramos los modelos exactos de ambos sólidos al no limitar el nivel máximo del árbol en la construcción del modelo.

3.7. Complejidad espacial

Debido a la similitud del esquema con Octrees Extendidos, en cuanto a la forma de codificar la información geométrica y la estructura jerárquica utilizada, en este apartado nos vamos a centrar en la comparación del espacio necesario para almacenar un modelo en el esquema propuesto con el necesario para hacerlo en el esquema Octrees Extendidos.

El hecho de utilizar nodos internos que almacenan parte de la información frontera que aparece en los nodos terminales de los Octrees Extendidos, hace que la duplicidad de información que puede aparecer en nodos terminales vecinos se vea reducida.

En la figura 3.20 se muestra la representación mediante Octree Extendido (centro) y SP-Octree (derecha) del sólido de la izquierda. Como podemos observar en el detalle de una zona concreta del sólido, aparece una cara triangular muy pequeña junto a caras vecinas de gran tamaño (figura 3.20.a).

Esto hace que en el Octree Extendido tengamos que subdividir varias veces los nodos que contienen los tres vértices hasta que conseguimos que cada uno de ellos pertenezca a nodos distintos. Por tanto, las caras vecinas grandes se dividen y se repiten en los nodos terminales que van apareciendo al dividir los nodos grises que contienen la cara pequeña (figura 3.20.b).

Sin embargo, el modelo obtenido mediante SP-Octree necesita un árbol de sólo dos niveles de profundidad, donde esa cara pequeña junto con las vecinas forman una convexidad, de forma que pueden almacenarse en un único nodo convexo o, como ocurre en este caso, en el nodo gris del árbol, por lo que no es necesario repetirlas en los nodos hijo (figura 3.20.c).

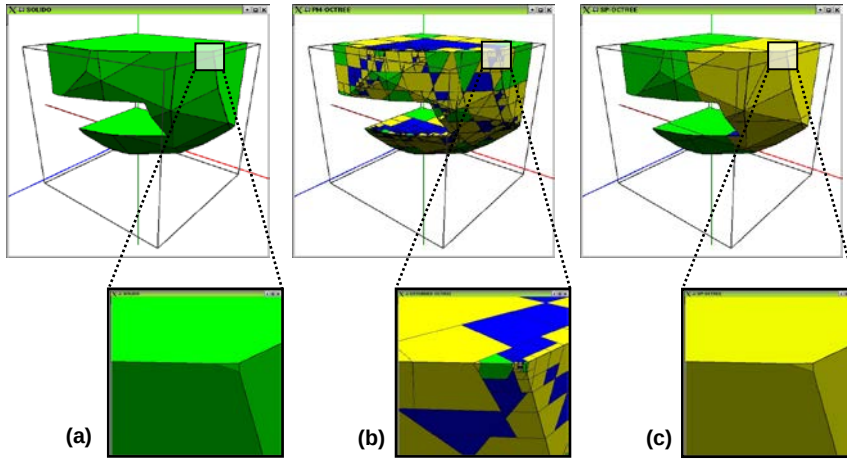


Figura 3.20. Sólido B-Rep (a) representado con *Octree Extendido* (b) y *SP-Octree* (c).

Además, debemos tener en cuenta que en ambos esquemas de representación el tipo de nodo vértice es el que más espacio de almacenamiento necesita (ver apartado 3.4) por tener que codificar, además de las referencias a los planos que aparecen en el nodo, la configuración de cada una de las aristas que convergen en el vértice.

Como podemos ver a partir de la definición del tipo de nodo vértice en el esquema propuesto, el número de nodos vértice en un SP-Octree depende de la configuración de los vértices del sólido a representar (aristas cóncavas y convexas que convergen en él).

El número de nodos vértice que aparecen en un Octree Extendido es siempre el número de vértices del sólido representado. Sin embargo, en el modelo propuesto sólo se almacenan como nodos vértice aquellos que presentan una configuración especial. Por tanto, el modelo propuesto será generalmente más conciso.

En el siguiente apartado veremos con más detalle una serie de modelos utilizados para comparar el espacio de almacenamiento necesario entre el modelo propuesto y los Octrees Extendidos.

3.7.1. Evaluación del espacio de almacenamiento

Para estudiar el espacio necesario en los dos esquemas de representación vamos a comparar el número de nodos de cada tipo que aparece en cada esquema, así como el tamaño total de espacio necesario para cada modelo (en bytes).

Como el número de planos de cada modelo es el mismo en los dos esquemas y se utiliza la misma estructura para almacenar esa información geométrica, la diferencia en el espacio ocupado aparecerá únicamente en el almacenamiento de la estructura jerárquica.

La tabla 3.3 muestra los sólidos utilizados para las pruebas realizadas, ordenados por número de caras del objeto, así como los modelos resultantes tanto en Octree Extendido como SP-Octree.

Para cada objeto en la primera columna vemos el identificador con el que referenciamos a cada modelo en las tablas siguientes, una imagen del sólido representado, el número de caras y vértices del sólido y la profundidad de los árboles obtenidos para Octrees Extendidos y para SP-Octree.

Mientras que para visualizar los modelos SP-Octrees se ha utilizado la codificación de color descrita en el apartado 2.6, en los modelos Octrees Extendidos se ha utilizado el siguiente código de color para distinguir los distintos tipos de nodos terminales: ***verde*** para nodos *vértice*, ***azul*** para nodos *cara y* ***amarillo*** para los nodos *arista.*

Para realizar la comparación entre los dos esquemas, en los Octrees Extendidos sólo se han tenido en cuenta estos tres tipos de nodos terminales. No se contempla la utilización de nodos ***casi-vértice*** que incluiría dos o más caras que comparten un vértice en común aunque sea fuera del nodo.

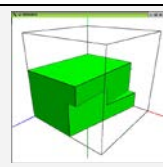
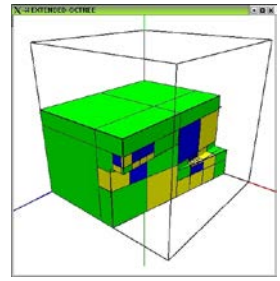
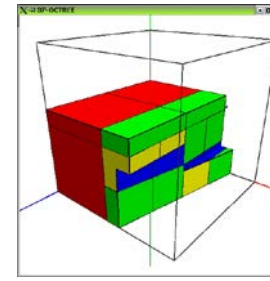
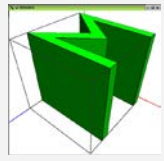
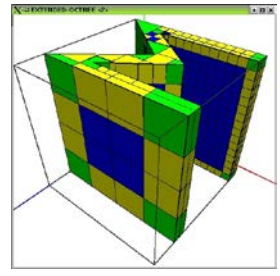
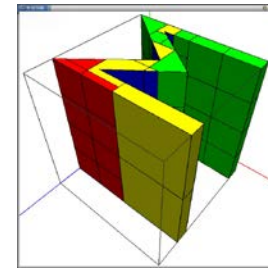
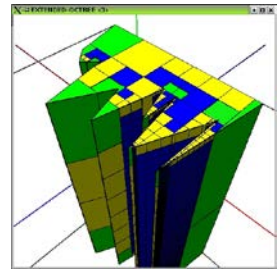
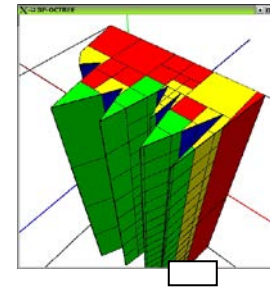
Sólido	*Octree Extendido*	*SP-Octree*
1  Número de caras: 9 Número de vértices: 14 Profundidad OE: 9 Profundidad SPO: 3		
2  Número de caras: 12 Número de vértices: 20 Profundidad OE: 6 Profundidad SPO: 3		
3  Número de caras: 16 Número de vértices: 28 Profundidad OE: 8 Profundidad SPO: 5		

Tabla 3.3. Modelos utilizados para evaluación del espacio ocupado (1/5).

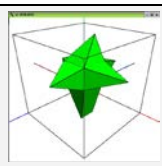
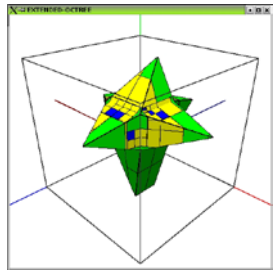
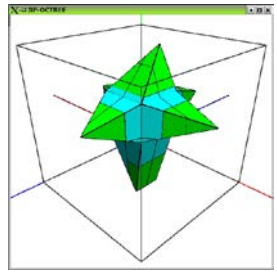
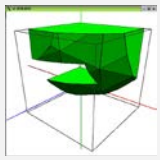
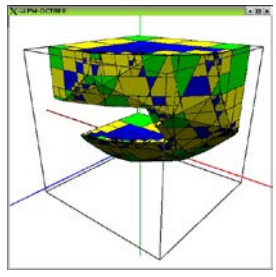
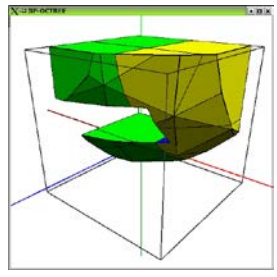
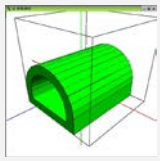
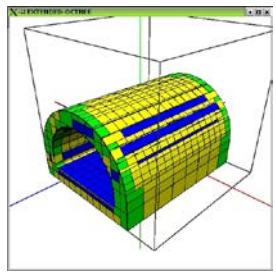
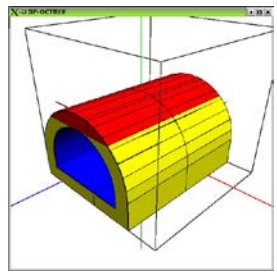
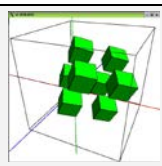
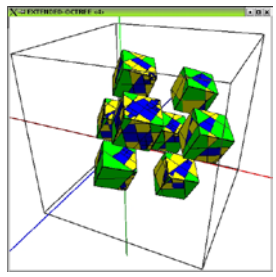
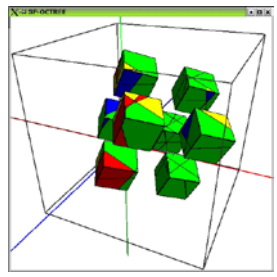
Sólido	*Octree Extendido*	*SP-Octree*
4  Número de caras: 24 Número de vértices: 15 Profundidad OE: 5 Profundidad SPO: 2		
5  Número de caras: 30 Número de vértices: 45 Profundidad OE: 11 Profundidad SPO: 1		
6  Número de caras: 40 Número de vértices: 76 Profundidad OE: 5 Profundidad SPO: 1		
7  Número de caras: 48 Número de vértices: 64 Profundidad OE: 11 Profundidad SPO: 3		

Tabla 3.3. Modelos utilizados para evaluación del espacio ocupado (2/5).

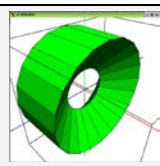
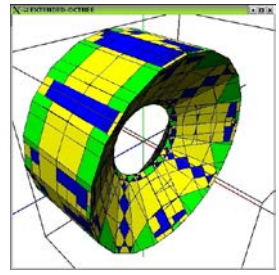
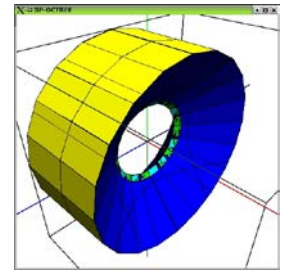
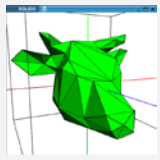
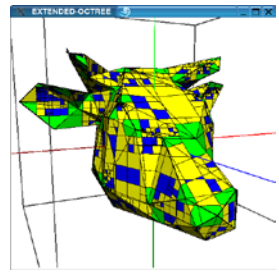
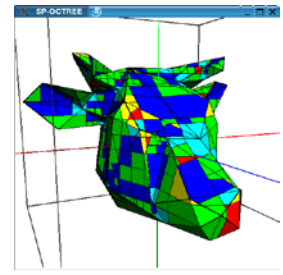
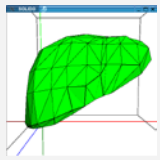
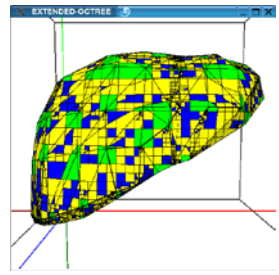
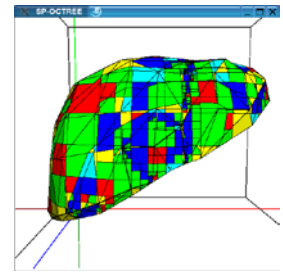
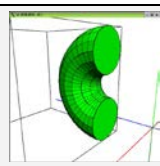
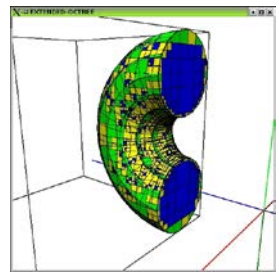
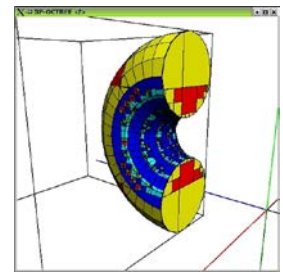
Sólido	*Octree Extendido*	*SP-Octree*
8  Número de caras: 60 Número de vértices: 60 Profundidad OE: 11 Profundidad SPO: 11		
9  Número de caras: 140 Número de vértices: 72 Profundidad OE: 15 Profundidad SPO: 10		
10  Número de caras: 274 Número de vértices: 139 Profundidad OE: 14 Profundidad SPO: 11		
11  Número de caras: 322 Número de vértices: 336 Profundidad OE: 12 Profundidad SPO: 9		

Tabla 3.3. Modelos utilizados para evaluación del espacio ocupado (3/5).

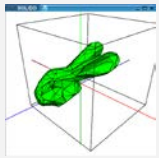
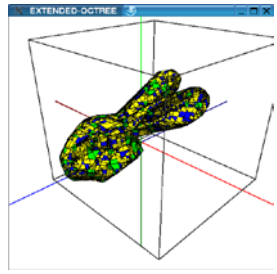
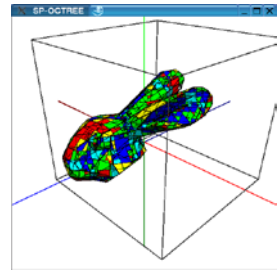
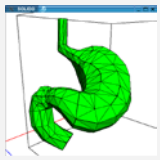
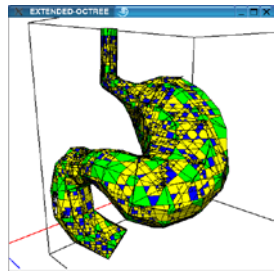
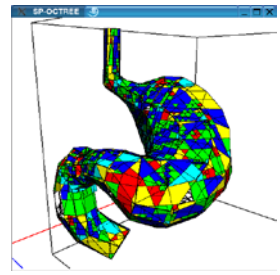
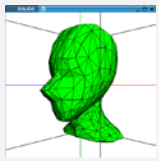
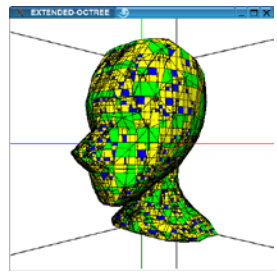
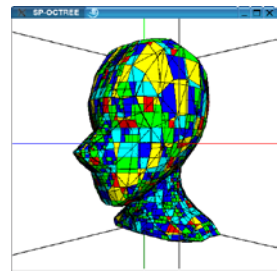
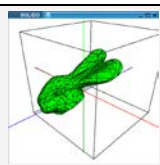
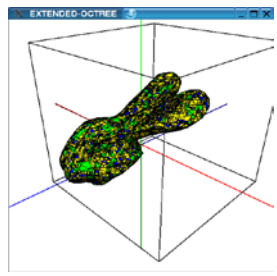
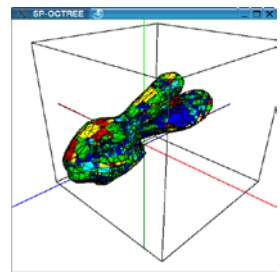
Sólido	*Octree Extendido*	*SP-Octree*
12  Número de caras: 404 Número de vértices: 204 Profundidad OE: 15 Profundidad SPO: 12		
13  Número de caras: 534 Número de vértices: 269 Profundidad OE: 16 Profundidad SPO: 17		
14  Número de caras: 796 Número de vértices: 401 Profundidad OE: 16 Profundidad SPO: 13		
15  Número de caras: 812 Número de vértices: 408 Profundidad OE: 18 Profundidad SPO: 16		

Tabla 3.3. Modelos utilizados para evaluación del espacio ocupado (4/5).

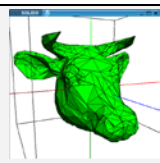
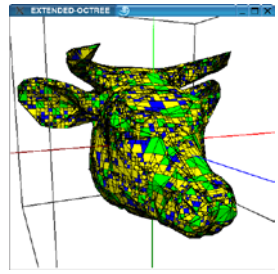
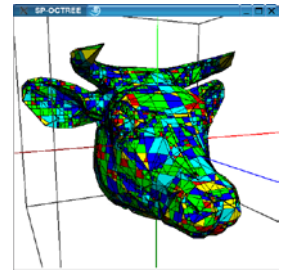
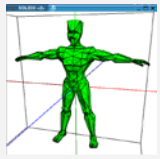
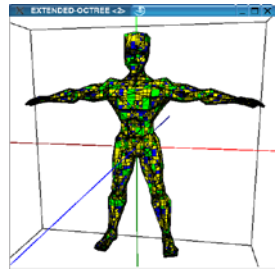
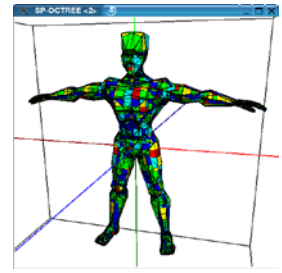
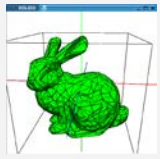
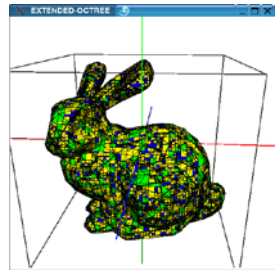
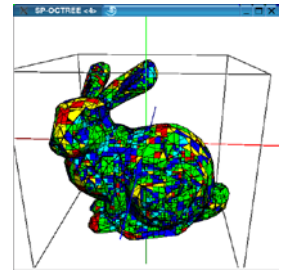
Sólido	*Octree Extendido*	*SP-Octree*
16  Número de caras: 1076 Número de vértices: 540 Profundidad OE: 16 Profundidad SPO: 16		
17  Número de caras: 1376 Número de vértices: 714 Profundidad OE: 17 Profundidad SPO: 16		
18  Número de caras: 1466 Número de vértices: 735 Profundidad OE: 16 Profundidad SPO: 14		

Tabla 3.3. Sólidos utilizados para evaluación del espacio ocupado (5/5).

La tabla 3.4 muestra los resultados en cuanto a profundidad del árbol y al número de nodos por tipo para los modelos Octrees Extendidos. Como podemos ver, el número de nodos *vértice* aumenta con la complejidad del objeto, ya que coincide con el número de vértices del sólido.

			OCTREE EXTENDIDO							
			Profundidad	NODOS						
Modelo	Planos	Vértices		Blancos	Negros	Grises	Cara	Arista	Vértice	Total
1	9	14	9	35	33	27	49	59	14	217
2	12	20	6	136	88	161	728	156	20	1289
3	16	28	8	596	696	1317	7438	462	28	10537
4	24	15	5	85	17	31	31	70	15	249
5	30	45	11	320	184	180	430	283	45	1442
6	40	76	5	340	83	234	562	569	76	1864
7	48	64	11	166	350	194	579	200	64	1553
8	60	60	11	200	374	252	712	419	60	2017
9	140	72	15	1351	1526	1005	2446	1641	72	8041
10	274	139	14	2778	2799	1878	3412	4019	139	15025
11	322	336	12	1010	876	613	827	1244	336	4906
12	404	204	15	6050	5171	3688	6459	7933	204	29505
13	534	269	16	4467	3839	2690	3946	6310	269	21521
14	796	401	16	5386	4644	3075	3678	7417	401	24601
15	812	408	18	10188	8629	6759	15845	12239	408	54068
16	1076	540	16	11619	10446	7434	14384	15043	540	59466
17	1376	714	17	18228	15596	10743	19810	10856	714	75947
18	1466	735	16	14619	13234	8900	13534	20178	735	71200

Tabla 3.4. *Octrees Extendidos*: profundidad y nodos.

Además, la división de un nodo viene dado por el número de vértices, aristas o caras que aparecen en el nodo, mientras que en el modelo SP-Octree no depende del número de aristas y caras que aparezcan en él, sino de la configuración de las aristas que aparecen en él.

La tabla 3.5 muestra los mismos datos para los modelos SP-Octree. Como vemos el nivel de profundidad de los árboles SP-Octree es normalmente menor que para los correspondientes Octrees Extendidos.

El número de nodos *vértice* se reduce drásticamente, debido a que en la mayoría de los casos los nodos con un único vértice se clasifican como *grises*, almacenando en él algunas de las caras que comparten el vértice, y el resto de caras que forman concavidades se almacenan en los nodos hijo de ese nodo. Además, los vértices que aparecen en la intersección de planos convexos quedarán representados con nodos *convexos*, y los que aparezcan en la intersección de planos cóncavos se representarán mediante nodos *cóncavos.*

El número de nodos del resto de tipos comunes también disminuye en todos los casos, debido a que en SP-Octrees los nodos *cóncavos* y *convexos* pueden almacenar más de dos caras, por lo que pueden agrupar varios nodos terminales *arista*, *cara* y *vértice* del modelo Octree Extendido.

De igual forma, al reducir el número de niveles del árbol, el número de nodos totalmente fuera o dentro del sólido también disminuye considerablemente.

Aunque algunos de los nodos vértice de los árboles OE se han clasificado ahora como *grises*, el número final de ese tipo de nodos sigue siendo aún menor en los árboles SP-Octree que en los OE.

			SP-OCTREE							
			Profundidad	**NODOS**						
Modelo	**Planos**	**Vértices**		**Blancos**	**Negros**	**Grises**	**Cóncavos**	**Convexos**	**Vértice**	**Total**
1	9	14	3	8	6	6	8	21	0	49
2	12	20	3	5	0	11	18	55	0	89
3	16	28	5	52	102	103	288	280	0	825
4	24	15	2	28	0	7	0	14	8	57
5	30	45	1	1	0	1	3	4	0	9
6	40	76	1	0	0	1	4	4	0	9
7	48	64	3	44	0	18	6	77	0	145
8	60	60	11	46	123	76	139	205	20	609
9	140	72	10	439	212	224	330	550	38	1793
10	274	139	11	927	762	583	833	1488	72	4665
11	322	336	9	422	239	222	295	479	120	1777
12	404	204	12	1364	865	744	1065	1804	111	5953
13	534	269	17	1649	1263	956	1303	2328	150	7649
14	796	401	13	1846	1431	1029	1347	2336	244	8233
15	812	408	16	2212	1365	1265	1677	3376	226	10121
16	1076	540	16	4044	2823	2259	3035	5579	333	18073
17	1376	714	16	5026	3182	2892	3343	8285	409	23137
18	1466	735	14	3990	3650	2547	3598	6190	402	20377

Tabla 3.5. *SP-Octrees*: profundidad y nodos.

La tabla 3.6 muestra un resumen del total de nodos para cada uno de los modelos y el porcentaje de reducción obtenido al utilizar el esquema propuesto.

Como vemos se obtiene una reducción media de alrededor del 78% del número de nodos utilizados en Octrees Extendidos.

		OCTREE EXTENDIDO	*SP-OCTREE*	
Modelo	**Planos**	Total Nodos	Total Nodos	% Reducción
1	9	217	49	77,42%
2	12	1289	89	93,10%
3	16	10537	825	92,17%
4	24	249	57	77,11%
5	30	1442	9	99,38%
6	40	1864	9	99,52%
7	48	1553	145	90,66%
8	60	2017	609	69,81%
9	140	8041	1793	77,70%
10	274	15025	4665	68,95%
11	322	4906	1777	63,78%
12	404	29505	5953	79,82%
13	534	21521	7649	64,46%
14	796	24601	8233	66,53%
15	812	54068	10121	81,28%
16	1076	59466	18073	69,61%
17	1376	75947	23137	69,54%
18	1466	71200	20377	71,38%
			% medio de reducción	78,46%

Tabla 3.6. Reducción en el número de nodos totales.

Las figuras 3.21 y 3.22 muestran gráficamente la profundidad de los árboles generados y el número de nodos totales en Octree Extendidos y SP-Octrees para los modelos utilizados en la comparación (tablas 3.4 y 3.5).

Podemos ver que la profundidad obtenida en los árboles SP-Octree es, normalmente, menor que la necesaria para los Octrees Extendidos. Aún cuando la profundidad es igual o mayor en el nuevo esquema (modelos 8, 13 y 16), podemos observar en la figura 3.22 que el número de nodos totales que aparecen en el SP-Octree es menor que en el modelo Octree Extendido.

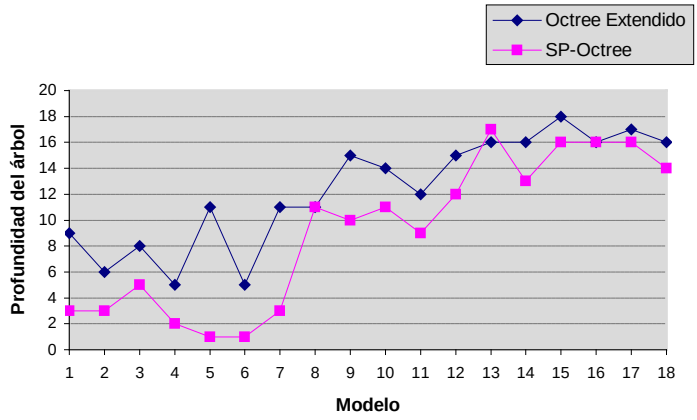


Figura 3.21. Profundidad de los árboles OE y SP-Octree.

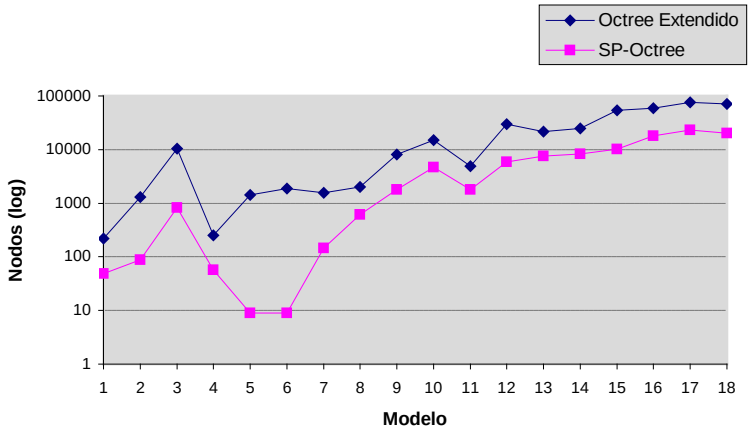


Figura 3.22. Nodos totales en OE y SP-Octree.

En la tabla 3.7 podemos ver el espacio, en bytes, utilizado para almacenar el modelo completo en cada uno de los esquemas. El número de planos es el mismo en los dos modelos, y al utilizar la misma codificación para la lista auxiliar, el tamaño utilizado es el mismo en los dos esquemas (tercera columna de la tabla).

Por tanto, la diferencia entre los dos modelos viene dada por el espacio necesario para almacenar el árbol, que como vemos es bastante menor en el caso de SP-Octrees, obteniéndose una reducción media alrededor del 60% respecto al espacio utilizado para almacenar los Octrees Extendidos.

			O. EXTENDIDO		*SP-OCTREE*		
Modelo	**Número de planos**	**PLANOS (bytes)**	**ARBOL (bytes)**	**TOTAL (bytes)**	**ARBOL (bytes)**	**TOTAL (bytes)**	**% Reducción**
1	9	445	1688	2133	377	822	61,46%
2	12	593	6902	7495	771	1364	81,80%
3	16	792	48351	49143	6161	6953	85,85%
4	24	1189	2408	3597	1132	2321	35,47%
5	30	1489	8651	10140	240	1729	82,95%
6	40	1988	15030	17018	350	2338	86,26%
7	48	2400	10450	12850	1219	3619	71,84%
8	60	2987	14512	17499	6187	9174	47,57%
9	140	7018	49522	56540	16841	23859	57,80%
10	274	13708	101353	115061	41129	54837	52,34%
11	322	16129	52076	68205	22110	38239	43,94%
12	404	20207	193874	214081	56423	76630	64,21%
13	534	26741	159654	186395	70644	97385	47,75%
14	796	40103	192842	232945	86299	126402	45,74%
15	812	40592	360962	401554	104090	144682	63,97%
16	1076	53908	418762	472670	174427	228335	51,69%
17	1376	68378	596100	664478	225882	294260	55,72%
18	1466	73683	532363	606046	208448	282131	53,45%
					% medio de reducción		60,54%

Tabla 3.7. Espacio de almacenamiento necesario para OE y SP-Octrees.

La figura 3.23 muestra la relación entre el tamaño del árbol para SP-Octree y para Octree Extendido como una dispersión de puntos en la que vemos cómo, para todos los modelos, los puntos se encuentran sobre la diagonal, reflejando el mayor espacio ocupado por los árboles OE.

En la figura 3.24 vemos gráficamente la reducción que conseguimos en espacio total necesario para almacenar cada uno de los modelos utilizados, en función del número de caras del sólido representado.

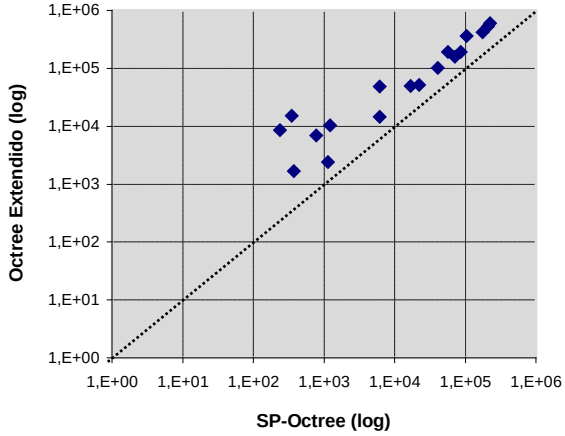


Figura 3.23. Relación del tamaño (bytes) de los árboles OE y SP-Octree.

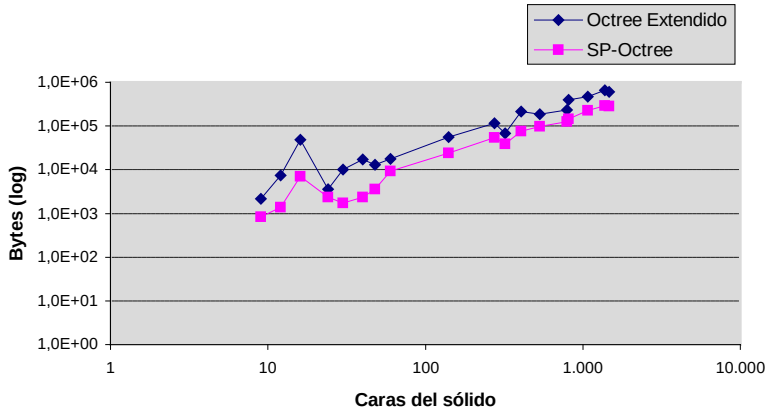


Figura 3.24. Tamaño (bytes) del modelo según número de caras del sólido.

Como puede observarse de las tablas 3.7 y 3.5, los sólidos para los que la diferencia de tamaño necesario para almacenar los modelos es menor de un 60% corresponden a los modelos SP-Octree en los que aparecen nodos vértice.

Esto se debe a que el número de vértices del sólido en los que convergen aristas cóncavas y convexas es mayor, y el tipo de nodo *vértice*, que almacena este tipo de geometría, es el que más espacio necesita para su representación al tener que almacenar la configuración de las aristas que aparecen en él (figura 3.25).

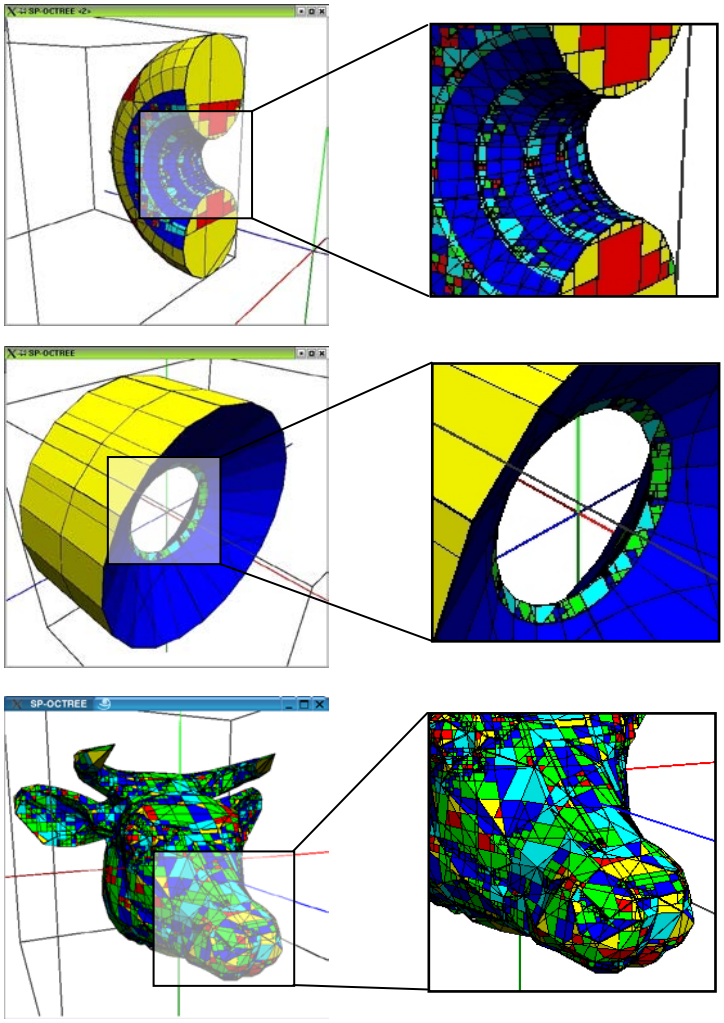


Figura 3.25. Nodos *vértice* - celeste- en SP-Octrees (detalle).

En cuanto a los que más diferencia presentan (más de un 80% de reducción), son los modelos 2,3, 5 y 6.

Como se puede observar en las figuras 3.26 y 3.27, se debe a que son objetos en los que aparecen caras paralelas o casi paralelas muy cercanas, o con polígonos muy pequeños donde los OE deben descender mucho en el árbol para conseguir separarlos, mientras que los SP-Octrees, al permitir ese tipo de configuraciones de forma directa son capaces de representarlas en los niveles iniciales del árbol.

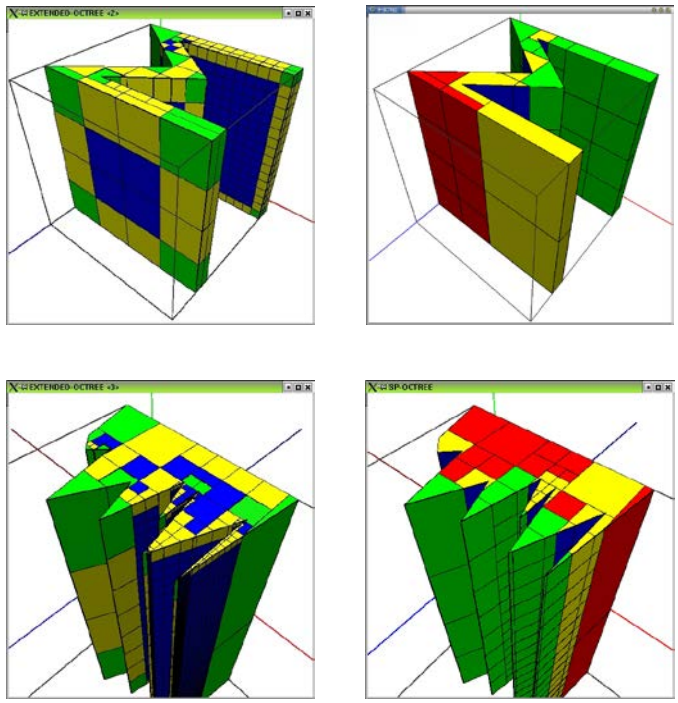


Figura 3.26. OE (izquierda) y SP-Octrees (derecha) para sólidos con caras paralelas o casi paralelas.

En la imagen 3.26 podemos ver, en color azul en el modelo Octree Extendido, los nodos *cara* creados en niveles muy profundos del árbol para poder separar los planos paralelos o muy cercanos. En esa misma imagen

(derecha) vemos que en el modelo SP-Octree dichos planos pertenecen a nodos *convexos* o *grises* en niveles iniciales.

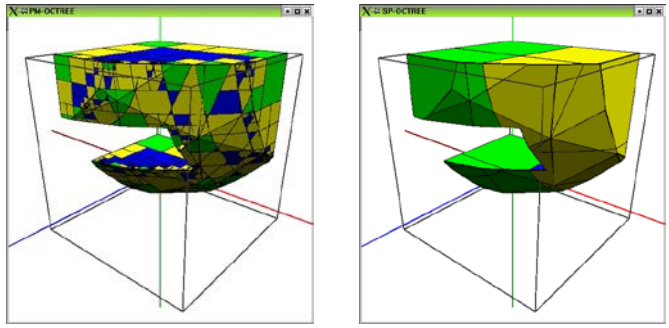


Figura 3.27. OE (izquierda) y SP-Octree (derecha) para sólido con caras muy pequeñas.

En la figura 3.28 podemos ver una última configuración en la que el número de niveles necesario aumenta en OE respecto a SP-Octree. Ocurre cuando aparecen caras definidas por muchos vértices, en cuyo caso el OE se debe subdividir hasta conseguir que cada uno de los vértices quede en un nodo.

Sin embargo, en SP-Octree esas caras aparecen almacenadas en el nodo raíz del árbol al pertenecer a la envolvente convexa del sólido.

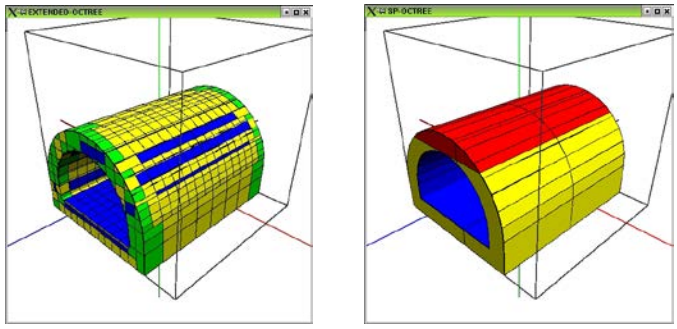


Figura 3.28. OE (izquierda) y SP-Octree (derecha) para sólido con caras de muchos vértices.

3.8. Propiedades del esquema de representación

El esquema propuesto hereda las propiedades básicas de los esquemas de representación de sólidos basados en descomposición espacial y son parecidas a las de los Octrees clásicos y extensiones propuestas a los mismos:

- *Dominio de representación:* al igual que las extensiones propuestas de los Octrees Clásicos que incluyen información adicional en los nodos terminales, los SP-Octrees permiten representar de forma exacta sólidos formados por caras planas.

  Como se comentó en el apartado 3.6 el esquema propuesto permite representar también objetos poliédricos formados por varios cuerpos separados e incluso objetos con agujeros.

  Sin embargo, al igual que ocurre en los Octrees Extendidos, con los tipos de nodos establecidos el esquema permite inicialmente representar sólo sólidos *2-variedad* [Mänt88]. Para poder representar objetos *no-variedad* generales necesitaríamos ampliar los tipos de nodos utilizados de forma similar a la propuesta para Octrees Extendidos [Brun90a].

- *Validez:* con los tipos de nodos definidos para el modelo es fácil comprobar la validez de una representación en el esquema propuesto.

- *Ambigüedad:* siempre que alcancemos el máximo nivel necesario, todo modelo SP-Octree representa un único sólido.

- *Unicidad:* Al igual que ocurre con los Octrees clásicos [Mänt88], podemos tener varias representaciones válidas de un modelo SP-Octree con distintas profundidades en el árbol (si no alcanzamos el máximo nivel necesario tendríamos sólo aproximaciones al objeto representado).

  Sin embargo, para un nivel definido, la representación de un sólido siempre será *única* si se cumplen las siguientes condiciones, establecidas por la propia definición de los tipos de nodos utilizados en el esquema:

  a) Los planos utilizados en un modelo están ordenados de forma que en cada nodo hay una forma única de referenciar los planos que aparecen en él.

b) Los planos aparecen siempre en el nivel del árbol más alto posible.

c) Todos los planos que aparecen en un nodo forman parte de la frontera o pertenece a la envolvente convexa de la parte del sólido representada en él. Es decir, no aparecen planos que no aportan ninguna información a la hora de definir el sólido representado en un nodo.

Como resultado de algunas operaciones sobre el modelo (como por ejemplo con las operaciones booleanas) puede ocurrir que el modelo resultante contenga planos que necesitemos eliminar o que deban agruparse para que se cumplan estas condiciones y podamos asegurar la unicidad del esquema. Por ello, es necesario realizar un postproceso tras realizar las mismas para *compactar* el árbol obtenido.

- *Concisión:* como se ha discutido en el apartado anterior, los modelos SP-Octrees permiten reducir la profundidad del árbol utilizado en la representación y evitan la repetición de planos en nodos vecinos. Esto hace que, en general, necesite menos espacio de almacenamiento que los Octrees Extendidos.

En los siguientes capítulos se discutirán con detalle algunos de los algoritmos básicos sobre el esquema que van a facilitar la visualización y el cálculo sobre el modelo, así como la creación del mismo a partir de otros esquemas de representación existentes.

Sin embargo, algunas de las operaciones que con Octrees clásicos son sencillas, en el nuevo esquema propuesto pierden su simplicidad al tener que operar con los planos que aparecen en cada nodo.

3.9. Conclusiones

En este capítulo se han presentado las bases sobre las que se ha definido el esquema de representación jerárquico *SP-Octree* basado en la inclusión de información de parte de la frontera del sólido representado en los nodos internos del árbol.

Se han presentado los distintos tipos de nodos terminales utilizados, se ha descrito la modificación incluida en los nodos internos del árbol y se ha definido la parte del sólido representada por cada uno de esos tipos de nodos.

También se ha mostrado la representación interna utilizada para almacenar la información geométrica necesaria para representar de forma exacta objetos poliédricos y la estructura jerárquica que soporta el modelo.

Se ha descrito el proceso de construcción del modelo a partir de un sólido definido por sus caras frontera y, para facilitar la comprensión del esquema propuesto, se han incluido una serie de ejemplos de sólidos poliédricos representados con el nuevo esquema en los que podemos ver los distintos tipos de nodos utilizados, así como distintas situaciones con las que nos podemos encontrar.

Por último, se ha estudiado la complejidad espacial necesaria para almacenar el modelo, comparándola con el esquema de representación Octree Extendido, y se han presentado las propiedades que presenta el nuevo esquema de representación propuesto.

CAPÍTULO

4

Operaciones básicas sobre el modelo

4

En este capítulo estudiaremos las operaciones básicas realizadas sobre sólidos representados con el esquema propuesto. Discutiremos el proceso de visualización del modelo, la conversión de/a otros modelos, así como algunas de las interrogaciones típicas que se pueden hacer sobre el mismo. En cada caso presentamos una evaluación del coste computacional y temporal de los algoritmos propuestos.

4.1. Introducción

Una vez descrito el esquema de representación propuesto y su representación interna dentro del ordenador, para completar la funcionalidad del mismo debemos diseñar e implementar las operaciones principales necesarias para trabajar sobre sólidos.

Las operaciones estudiadas e implementadas sobre el esquema han sido las siguientes:

- *Clasificación de elementos geométricos:*

  Una de las operaciones básicas sobre un sólido consiste en clasificar otros elementos para obtener aquellos que están en el interior, en el exterior o sobre la frontera del sólido.

  En nuestro caso nos hemos centrado exclusivamente en la clasificación de puntos, ya que esta es la operación más utilizada, y se usa como base para otras operaciones más complejas.

- *Visualización del modelo:*

  Una vez representado un sólido con un modelo dentro del ordenador, es necesario realizar una visualización del mismo que permita mostrarlo.

- *Reconstrucción de la frontera del sólido representado:*

  Para realizar el proceso de visualización del modelo, y para otras operaciones, es necesario obtener la frontera del sólido representado a partir de la información almacenada en el modelo.

- *Conversión de/a otros esquemas de representación de sólidos:*

  Normalmente los sistemas de Diseño Asistido por Ordenador que trabajan con objetos sólidos no trabajan con un único esquema de representación, sino que tratan de utilizar varios para obtener las ventajas de cada uno y evitar sus inconvenientes. De esta forma, el sistema utiliza en cada caso el esquema más apropiado en función de la operación a realizar sobre el modelo.

  Pero para ello es necesario ofrecer mecanismos que permitan convertir el modelo de un esquema a otro.

- *Operaciones booelanas entre modelos:*

  Las operaciones booelanas (intersección, unión y diferencia) son un método muy potente y a la vez sencillo para poder construir modelos más complejos a partir de otros más sencillos. Implementar estas operaciones sobre el esquema de representación ofrece un mecanismo muy potente de creación de modelos. En el siguiente capítulo estudiaremos estas operaciones sobre el esquema propuesto.

Al ser un esquema de representación jerárquico todas las operaciones se basan en recorrer el modelo con un determinado orden y realizar dichas operaciones sobre cada uno de los nodos del árbol.

En los siguientes apartados se describe cada uno de los algoritmos diseñados para realizar estas operaciones.

4.2. Clasificación de puntos

Dentro de la Geometría Computacional y de la Informática Gráfica, el test de inclusión de un punto en sólidos poliédricos es uno de los problemas fundamentales debido al número de aplicaciones que tiene.

Existen numerosas técnicas para realizar esa clasificación en sólidos poliédricos, cada una de ellas con eficiencia, complejidad en su implementación y requerimientos específicos, por lo que cada uno de los métodos es más apropiado para casos concretos [Kala82, Lane84, Horn89, Joan96].

Uno de esos métodos [Feit98] utiliza aritmética entera para comprobar la inclusión de puntos en base a una descomposición en tetraedros del sólido y la comprobación de los *signos* (que indican la pertenencia o no) del punto respecto a cada tetraedro. Este último algoritmo es muy robusto y estable, además de tener una implementación sencilla.

Casi todos los métodos están diseñados para trabajar con caras triangulares, por lo que si la malla poligonal que define la frontera del sólido no está formada por triángulos, es necesario incluir un preproceso para descomponer cada polígono de la malla. Esto, junto con la complejidad de la malla a tratar, hace que sea interesante mejorar el tiempo necesario para realizar la clasificación de puntos en modelos grandes.

La clasificación de puntos en sólidos representados mediante esquemas jerárquicos tiene la ventaja de utilizar la propia estructura en árbol como índice espacial que disminuye el número de casos a tratar. La utilización de Octrees Clásicos y árboles BSP como índices espaciales mejoran el proceso de clasificación de puntos al reducir en cada nodo el número de caras del sólido a tratar.

En nuestro esquema, parte de la información frontera del sólido se encuentra representada en los nodos internos del árbol. De esta forma, en cada nodo interno, además de la caja englobante del mismo, podemos utilizar los planos que aparecen en él y que definen la envolvente convexa del sólido en ese nodo para acelerar el proceso de clasificación.

El primer paso para determinar si un punto es interior o no al sólido representado por un SP-Octree es comprobar si pertenece a la caja englobante del nodo raíz. Si no es así, el punto es exterior al sólido (algoritmo 4.1).

```
FUNCTION ClasificaPunto (p, SPOctree)
{

  if (p ∉ BoundingBox(NodoRaiz(SP-Octree))
    p ∉ Sólido
  else
    ClasificaNodo(p, NodoRaiz(SPOctree));
}
```

Algoritmo 4.1. Inclusión de un punto *p* en un SP-Octree.

```
FUNCTION ClasificaNodo (p, Nodo)
{
  if (Nodo es GRIS)
    return ClasificaNodoGris(p,Nodo);
  else
    if (Nodo es BLANCO)
      return EXTERIOR
    else
      if (Nodo es NEGRO)
        return INTERIOR
      else
        if (Nodo es CONVEXO)
          if (p ∉ ∩Pk del Nodo)
            return EXTERIOR
          else
            if (p sobre Pk del Nodo)
              return FRONTERA
            else
              return INTERIOR
        else
          if (Nodo es CONCAVO)
            if (p ∈ ∩Pk del Nodo invertidos)
              return EXTERIOR
            else
              if (p sobre Pk del Nodo)
                return FRONTERA
              else
                return INTERIOR
          else
            if (Nodo es VERTICE)
              return ClasificaVertice(p, Nodo);
}
```

Algoritmo 4.2. Inclusión de un punto *p* en un nodo de un SP-Octree.

Si el punto pertenece a ella, debemos iniciar el recorrido del árbol, comprobando la inclusión del punto en cada nodo del mismo de forma recursiva, atendiendo al tipo de nodo que se trate en cada momento y a la información geométrica almacenada en cada uno de ellos. En el algoritmo 4.2 se describen los pasos principales de este proceso.

Al tener almacenadas en cada nodo una lista de semiespacios definidos por los planos a los que se referencia en el nodo, todas las comprobaciones de pertenencia en el algoritmo clasifican el punto como interior, exterior o sobre cada uno de esos planos. Esto permite directamente saber si el punto es interior, exterior o pertenece a la frontera del sólido.

Si el nodo es *gris* se comprueba si el punto está en el exterior de todos los semiespacios definidos por los planos del nodo, en cuyo caso el punto no pertenece al sólido. Si está en el interior de todos ellos, se calcula a cuál de los ocho hijos pertenece el punto, realizando la llamada recursiva al algoritmo con dicho nodo.

Si el punto está sobre alguno de los planos que aparecen en un nodo *gris*, no podemos asegurar que se encuentre sobre la frontera del sólido ya que podría estar fuera del mismo. En este caso, debemos seguir descendiendo en la estructura de forma que si el punto finalmente es clasificado como exterior, entonces no pertenece al sólido. En otro caso, el punto pertenece a su frontera (algoritmo 4.3).

La figura 4.1 muestra un ejemplo (en 2D) de nodo *gris* que contiene los planos *A* y *B* del sólido representado, con los que se puede directamente clasificar el punto P_0 como exterior al sólido. Sin embargo, para clasificar el resto de puntos debemos descender en el árbol.

En este ejemplo el punto P_1 es interior a los dos planos del nodo *gris*, pero al clasificarlo respecto al hijo 2 al que pertenece, es exterior al plano *C* del nodo *convexo*, por lo que el punto no pertenece al sólido.

El punto P_2 es interior en el nodo *gris* y pertenece al hijo 0 que es *negro*, por lo que el punto es interior al sólido.

El punto P_3 está sobre el plano *B* del nodo *gris*, pero al clasificarlo respecto al nodo hijo 3, el punto es exterior a los dos planos C y D de ese nodo *cóncavo*, por lo que el punto no pertenece al sólido.

Por último, el punto P_4 está sobre el plano *A* del nodo *gris*, y al clasificarlo en el nodo *cóncavo* 2 que contiene al plano *C*, es interior a ese plano por lo que el punto está sobre la frontera del sólido.

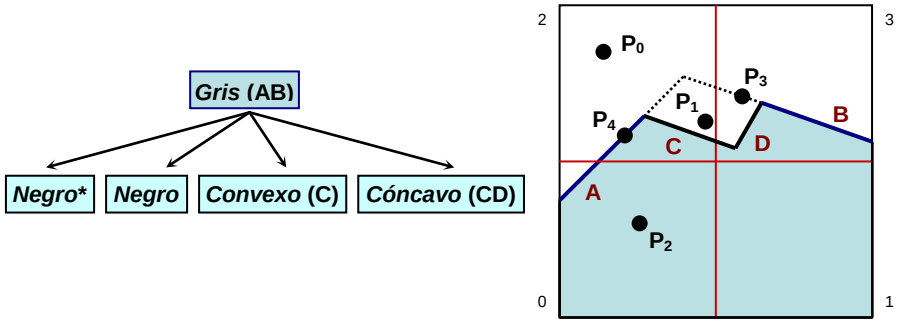


Figura 4.1. Inclusión de puntos en nodo *gris* de un SP-Octree.

```
FUNCTION ClasificaNodoGris (p, Nodo)
{
  if (p ∉ ∩Pk del Nodo)
    return EXTERIOR
  else
  {
    i= CalculaHijo(p, Nodo);
    Situación= ClasificaNodo(p, Nodo[i]);

    if (p sobre Pk de Nodo)
      if (Situación ¡= EXTERIOR)
        return FRONTERA
      else
        return EXTERIOR
    else
      return Situación
  }
}
```

Algoritmo 4.3. Inclusión de un punto *p* en un nodo *Gris*.

La clasificación de un punto dentro de nodos *blancos* y *negros* es directa: si el nodo es *blanco,* el punto es exterior al sólido; si el nodo es *negro,* el punto está sobre la frontera del sólido si lo está sobre alguno de los planos que aparecen en sus nodos ancestros, o será interior al sólido si es interior respecto a esos nodos.

La pertenencia a nodos *cóncavos* y *convexos* se obtiene con la comprobación de la pertenencia del punto al interior, exterior o la frontera de cada plano P_k incluido en el nodo.

En el caso de nodo *convexo,* si el punto es interior a todos los semiespacios definidos por los planos referenciados en el nodo, el punto será interior al sólido. Si está sobre alguno de esos planos y dentro del resto, el punto pertenecerá a la frontera del sólido, y si está fuera de alguno de ellos, será exterior al sólido.

Dentro de un nodo *cóncavo,* si el punto es exterior a todos los semiespacios pertenecientes al nodo, entonces no pertenece al sólido. Si está sobre alguno de los planos y fuera del resto, el punto pertenece a la frontera del sólido. En otro caso pertenecerá al interior del sólido.

Por último, en el nodo vértice aparece el caso más complejo a tratar. Debido a las múltiples configuraciones que pueden aparecer en esos nodos, a partir de la información de los planos que aparecen en él y de la configuración de las aristas, se reconstruye la geometría de la parte del sólido representada en el nodo. A partir de dicha geometría se realiza la clasificación del punto con cualquiera de los métodos existentes mediante la función *ClasificaVertice(p,Nodo)*.

4.2.1. Evaluación del algoritmo

Para evaluar el algoritmo de inclusión de puntos se ha comparado con el método propuesto en [Feit98] y con un algoritmo similar de inclusión de puntos en Octrees Extendidos, utilizando sólidos con distinta complejidad.

Como comentamos en el apartado anterior, este método necesita como entrada una malla de triángulos, por lo que antes de realizar la clasificación de puntos respecto a un sólido, la frontera del mismo debe descomponerse en triángulos. Para la comparación realizada no se ha tenido en cuenta el tiempo necesario para realizar esa triangulación.

La evaluación del método se ha realizado bombardeando los modelos con 10.000, 100.000 y 1.000.000 de puntos generados de forma aleatoria dentro de la caja envolvente de los sólidos.

En la figura 4.2 vemos algunos de los modelos utilizados en la evaluación del algoritmo. De algunos de ellos se han utilizado varias versiones con distinta resolución para estudiar la dependencia de los métodos estudiados en cuanto al número de caras del sólido representado.

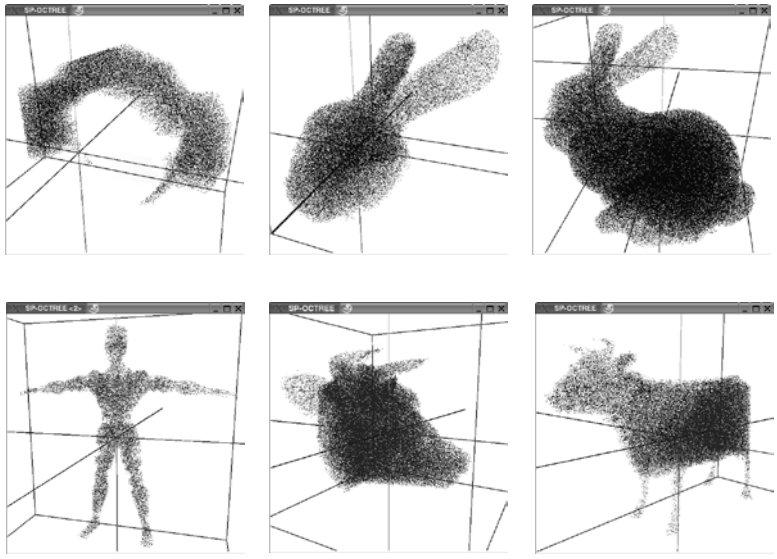


Figura 4.2. Bombardeo de puntos en los modelos utilizados en la evaluación

En nuestro caso, el algoritmo propuesto para clasificar puntos en un SP-Octree no necesita trabajar con triángulos, sino que se basa en la orientación de los puntos respecto a los planos almacenados en cada nodo del árbol dependiendo de su tipo. Esta misma idea ha sido la utilizada para implementar el algoritmo de inclusión de puntos sobre el modelo Octree Extendido.

La tabla 4.1 muestra los resultados de la comparativa de tiempos para el algoritmo propuesto utilizando SP-Octrees, Octrees Extendidos y el método propuesto en [Feit98] para mallas de triángulos. En este último caso no se ha tenido en cuenta el tiempo necesario para triangular la malla poligonal que define el sólido.

			SP-Octree		**Octree Extendido**		**B-Rep [Feit98]**
ID	Modelo	Planos	Profundidad del árbol	Tiempo (miliseg.)	Profundidad del árbol	Tiempo (miliseg.)	Tiempo (miliseg.)
1	Cabeza de conejo 5%	34	10	0,0025	13	0,0038	0,1451
2	Pieza mecánica	43	6	0,0023	15	0,0026	0,3741
3	Cabeza de conejo 10%	78	10	0,0026	15	0,0026	0,3586
4	Cabeza de conejo 15%	118	10	0,0030	17	0,0025	0,5311
5	Cabeza de conejo 20%	158	11	0,0029	17	0,0023	0,7059
6	Cabeza de conejo 25%	200	9	0,0030	14	0,0024	0,8935
7	Cabeza de conejo 50%	404	12	0,0032	15	0,0021	1,8584
8	Cabeza de vaca	424	17	0,0033	17	0,0049	2,7585
9	Vaca	590	13	0,0024	16	0,0044	3,2591
10	Cabeza de conejo 75%	608	15	0,0034	17	0,0022	2,7536
11	Conejo 25%	742	13	0,0044	18	0,0039	4,4584
12	Cabeza	796	13	0,0048	16	0,0030	4,7336
13	Cabeza de conejo 100%	812	16	0,0039	18	0,0021	3,6812
14	Cuerpo	1376	16	0,0043	17	0,0029	7,6418
15	Conejo 50%	1466	14	0,0054	16	0,0029	8,7409

Tabla 4.1. Tiempos (milisegundos) para test de inclusión de un punto.

En la figura 4.3 podemos observar gráficamente los resultados (en escala logarítmica) obtenidos para el test de intersección de un punto para cada uno de los modelos utilizados en la evaluación. Vemos claramente que el tiempo empleado por el algoritmo propuesto es siempre menor que para el algoritmo sobre la malla de triángulos.

Sin embargo, aplicando el mismo algoritmo sobre los Octrees Extendidos, aún cuando la profundidad del árbol en este esquema es generalmente mayor, el tiempo necesario para el test de inclusión es menor que para SP-Octree. Esto se

debe a que, aunque el número de nodos en los árboles SP-Octrees es siempre menor que en los OE, la complejidad de los nodos es mayor (número de planos sobre los que comprobar la inclusión en cada nodo).

Además, aunque muchos planos aparecen en nuestro esquema en los nodos superiores, estos permiten discriminar puntos exteriores al sólido, pero para comprobar si un punto es interior debemos descender siempre hasta los nodos terminales.

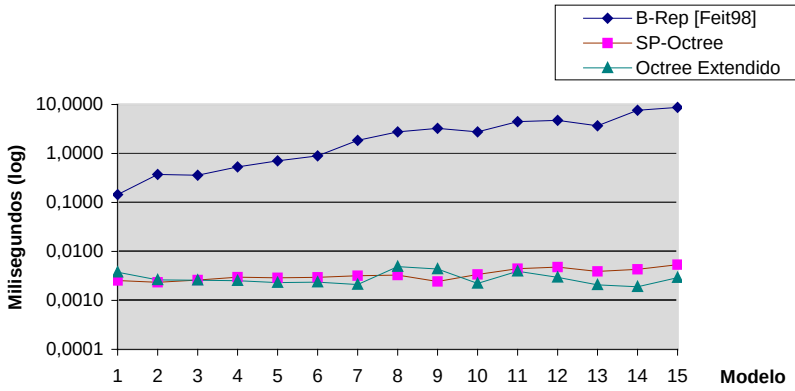


Figura 4.3. Tiempos para clasificación de un punto.

La figura 4.4 muestra estos mismos tiempos para los tres modelos en función del número de caras del modelo. Como vemos, la clasificación a partir de la malla triangular [Feit98] depende directamente del tamaño de la misma. Sin embargo, en el caso de SP-Octrees y OE, el tiempo necesario no depende del número de planos del modelo.

Como el algoritmo se basa en recorrer el árbol buscando a qué nodo pertenece el punto y clasificándolo en función del tipo del nodo y de la información que aparece en él y en sus ancestros, el tiempo necesario para el algoritmo depende ahora del número total de nodos del árbol (figura 4.5).

Además, dada la mayor complejidad del proceso de clasificación dentro de los nodos *vértice*, un aumento en el número de nodos de ese tipo implica un mayor coste en el tiempo necesario (figura 4.6).

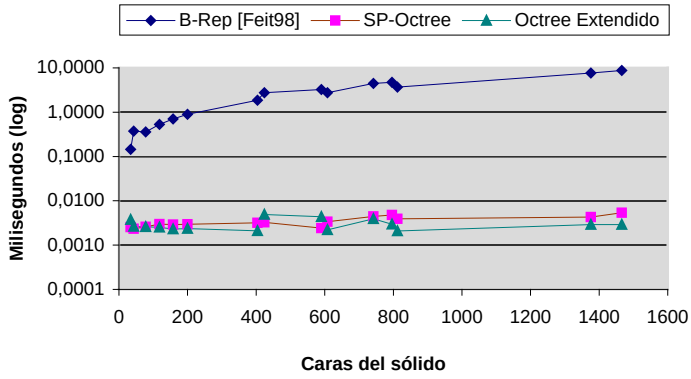


Figura 4.4. Clasificación de un punto según tamaño del sólido.

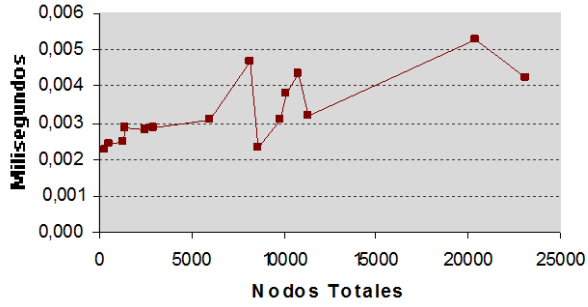


Figura 4.5. Clasificación de un punto en SP-Octree según nodos totales.

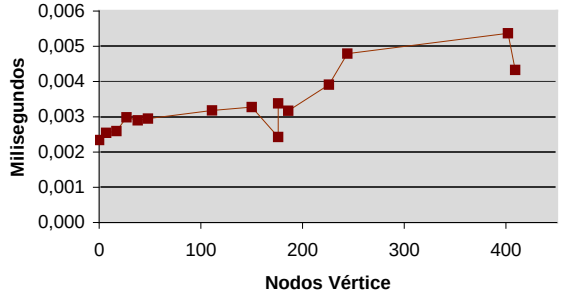


Figura 4.6. Clasificación de un punto en SP-Octree según nodos *vértice*.

4.3. Visualización del modelo

Una de las ventajas de los Octrees Clásicos es el orden implícito que se puede establecer entre los nodos, lo que facilita el proceso de visualización del modelo definiendo el orden de representación de los nodos. En el esquema que proponemos se mantiene ese orden implícito entre los nodos del árbol.

Con esta idea, y siguiendo el mismo algoritmo de visualización que en Octrees Clásicos, para visualizar un modelo en el esquema SP-Octree recorremos los nodos del árbol en un orden determinado, representando para cada nodo la información geométrica almacenada en él.

Al no almacenar información geométrica de la frontera explícitamente en los nodos, al igual que ocurre con los Octrees Extendidos, el proceso de visualización de cada nodo es similar al realizado para representarlo en OE.

Al visualizar un nodo, debemos recortar cada plano con la propia caja englobante del nodo y con los otros planos pertenecientes al mismo. Además, también debemos tener en cuenta los planos que aparece en sus ancestros y que, por tanto, forman parte de la envolvente del sólido. Por ello, también debemos recortar con esos planos el resultado obtenido anteriormente para obtener el polígono final.

El hecho de mantener información de parte de la frontera en los nodos internos permite realizar una visualización aproximada del sólido, simplemente limitando el nivel de profundidad máximo que queremos representar, haciendo que los nodos grises de ese nivel se visualicen como si se trataran de nodos *convexos*, representándolos por tanto de forma aproximada.

Esto permite obtener una visualización por niveles del modelo, proceso que servirá de base para realizar la transmisión progresiva del modelo como veremos en el capítulo 6 de esta memoria.

En la figura 4.7 vemos un ejemplo de visualización de un modelo limitando el nivel máximo representado. En color rojo representamos los nodos grises del nivel máximo visualizado en cada caso. Como podemos observar, cada uno de esos nodos está representado como un nodo *convexo* cuya geometría es una envolvente de la parte del sólido representado en el mismo, definida por los planos que aparecen en esos nodos *grises*.

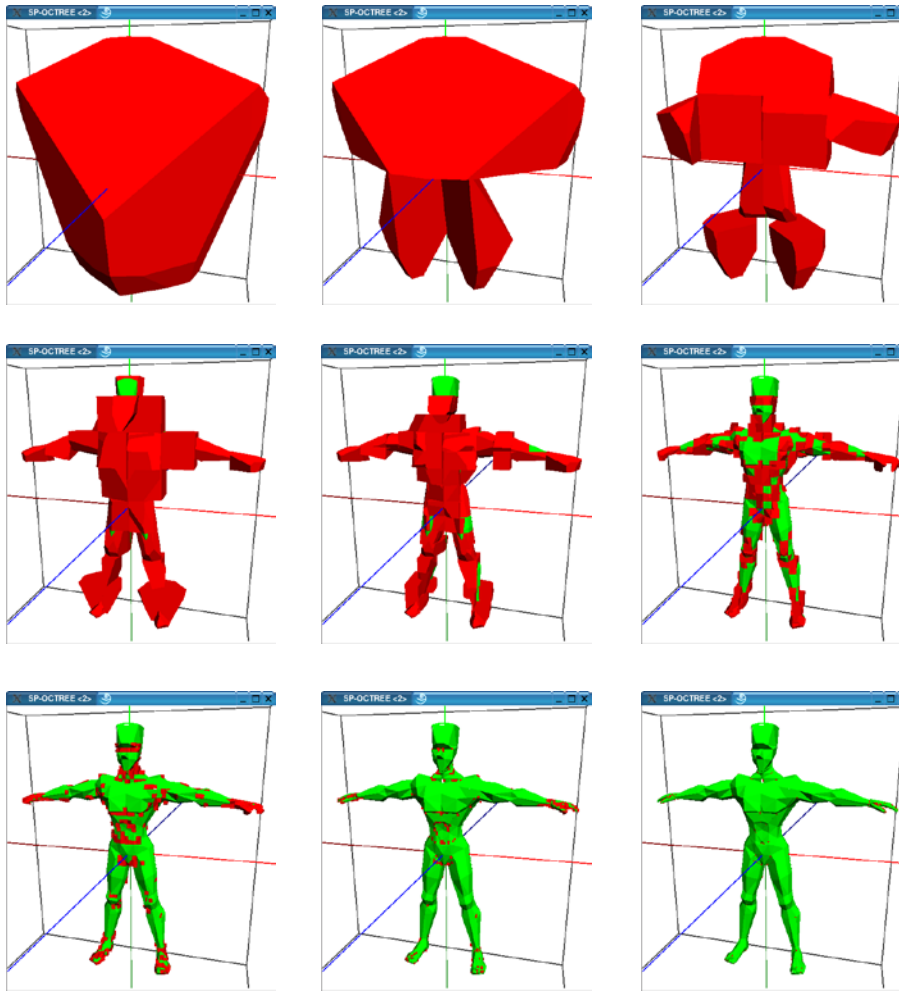


Figura 4.7. Visualización de modelo SP-Octree variando el nivel de profundidad máximo

El algoritmo 4.4 muestra los pasos iniciales para visualizar un modelo SP-Octree hasta un determinado nivel de profundidad, *NivelMax*, realizando un recorrido del árbol en preorden.

```
FUNCTION VisualizaSPO (SPO.Nodo, NivelMax, Ancestros)
{
  if (SPO.Nodo es Terminal)
    VisualizaTerminal (SPO.Nodo, Ancestros)
  else
    if (NivelMax==0)
      VisualizaGrisAproximado (SPO.Nodo, Ancestros)
    else
    {
      Insertar en Ancestros los planos Pk ∈ SPO.NodoRaiz;
      for (i=0; i<8; i++)
        VisualizaSPO (SPO.Nodo.Hijo[i], NivelMax-1, Ancestros)
    }
}
```

Algoritmo 4.4. Visualización de un modelo SP-Octree.

Este algoritmo recibe como entrada el nodo raíz de un SP-Octree, el nivel máximo a visualizar y una lista de planos perteneciente a los ancestros del nodo, que inicialmente estará vacía. Si ese nodo raíz del árbol es un nodo terminal (deberá ser un nodo *convexo*), se llama directamente a la función encargada de visualizar un nodo terminal.

Si se trata de un nodo *gris*, en caso de haber alcanzado el máximo nivel a visualizar, procedemos a visualizar dicho nodo como si se tratara de un terminal *convexo.* Esto lo realizamos con la llamada a la función *VisualizaGrisAproximado* que visualiza los planos del nodo *gris* recortándolos entre ellos y con los planos que aparecen en la lista *Ancestros.*

Pero debemos tener en cuenta que un nodo *gris* visualizado como *convexo* hará que los planos no tengan continuidad en los nodos vecinos, por lo que se producirían agujeros en la malla representada.

Aunque esto puede resolverse utilizando cualquiera de los algoritmos propuestos para eliminación de esos mismos errores en el proceso de visualización de isosuperficies mediante *Marching Cubes* [Lore87, Shek96, Vela02], actualmente hemos optado por resolver el problema de una forma mucho más simple: cerrar esos agujeros mediante los polígonos resultantes sobre las caras de la caja englobante del nodo.

En la figura 4.8 podemos ver en color gris los polígonos pertenecientes a planos en nodos *grises* de nivel 3. En color rosa vemos los polígonos sobre las caras de las cajas englobantes de los mismos, utilizados para cerrar los agujeros que se producen.

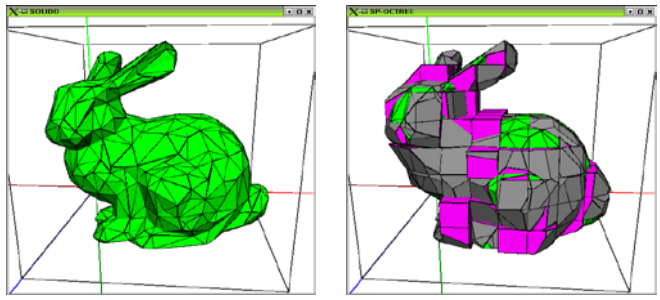


Figura 4.8. Aproximación de nodos *grises* de máximo nivel.

Si no estamos en el nivel máximo y el nodo es *gris*, añadimos los planos que aparecen en el nodo en la lista de planos *Ancestros* y procedemos a llamar a la función de forma recursiva para cada uno de los nodos hijo, decrementando el valor máximo de profundidad.

En el algoritmo 4.5 vemos el pseudocódigo para la visualización de un nodo terminal. Para cada plano en el nodo obtenemos el polígono resultante al recortarlos con la caja englobante y los planos convexos de los ancestros.

Para obtener el polígono a visualizar debemos recortar el resultado obtenido en función del tipo de nodo en el que nos encontremos:

- Si el nodo es *convexo*, basta con recortar dicho polígono con el resto de planos P_k en el nodo.
- Si el nodo es *cóncavo*, debemos recortarlo con el resto de planos pero con la normal invertida $\overline{P_j}$ para quedarnos con la zona del recorte exterior a esos planos, ya que forman concavidad con el polígono en cuestión.
- Si el nodo es *vértice*, recortamos el polígono con los planos vecinos, teniendo en cuenta la configuración (cóncava o convexa) de la arista que los une.

Con esto obtenemos el polígono resultante a visualizar para cada plano en el nodo. Aplicando un mecanismo de eliminación de caras ocultas entre todos los polígonos del nodo, y utilizando el orden adecuado de recorrido de los nodos en función de la posición del observador, obtenemos la correcta visualización del modelo.

```
FUNCTION VisualizaTerminal (SPO.Nodo, Ancestros)
{
  for (k=0; k<SPO.Nodo.NumPlanos; k++)
  {
    Poly = Recortar (Pk ∈ SPO.Nodo, BoundingBox(SPO.Nodo))

    for (i=0; i<Ancestros.NumPlanos; i++)
      Poly = Recortar (Poly, Pi ∈ Ancestros)

    case Tipo(SPO.Nodo)
    {
      CONVEXO: Poly = Recortar (Poly, Pj ∈ SPO.Nodo (j≠k))
               break;
      CONCAVO: Poly = Recortar (Poly, P̄j ∈ SPO.Nodo (j≠k))
               break;
      VERTICE: for (l=0; l < SPO.Nodo.NumAristas; l++)
               {
                 Config = Configuración (SPO.Nodo.Aristas[l])
                 Pj = Vecino (Pk, SPO.Nodo.Aristas[l])
                 Poly = Recortar (Poly, Pj, Config)
               }
               break;
    }

    Visualizar (Poly)
  }
}
```

Algoritmo 4.5. Visualización de nodo terminal de un SP-Octree

La figura 4.9 muestra el resultado del proceso de visualización descrito sobre algunos ejemplos de modelos SP-Octree, utilizando el código de color descrito en el capítulo 3:

- ***Verde*** para los planos pertenecientes a nodos *convexos*. Los planos pertenecientes a los ancestros de estos nodos también se representan en verde.

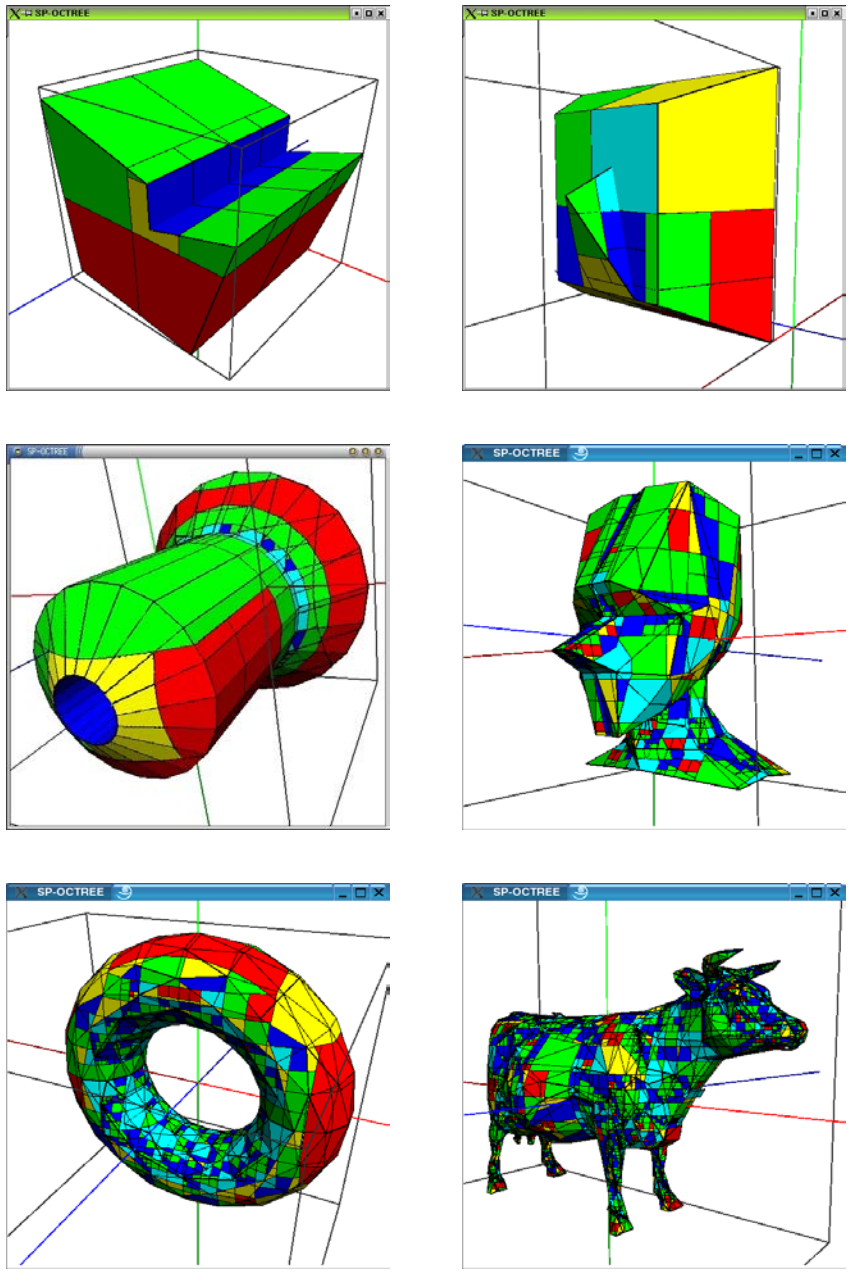


Figura 4.9. Visualización de modelos SP-Octree.

- ***Azul*** para los planos de los nodos *cóncavos*.
- ***Celeste*** para los nodos *vértice*.
- ***Amarillo*** para los planos que aparecen en los ancestros de un nodo *cóncavo* o *vértice*.
- ***Rojo*** para los planos existentes en los ancestros de nodos *negros*.

4.3.1. Evaluación

Debido a la similitud de los nodos utilizados en el esquema SP-Octree y los Octrees Extendidos, se ha implementado el mismo algoritmo de visualización en los dos esquemas para evaluar el tiempo empleado en cada uno.

En la tabla 4.2 mostramos el tiempo necesario para recorrer la estructura y obtener para cada nodo los polígonos a visualizar. La figura 4.10 muestra los sólidos utilizados para evaluar el algoritmo. Algunos de ellos son versiones con distinta resolución del mismo sólido con idea de utilizar una batería de pruebas que permita evaluar cómo afecta el número de caras originales del sólido en el proceso de visualización del modelo en ambos esquemas.

			Octree Extendido		**SP-Octree**	
Sólido	**Planos**	**Vértices**	**Nivel**	***Visualización (segundos)***	**Nivel**	***Visualización (segundos)***
1	30	45	0	0,05	7	0,08
2	40	76	11	0,09	11	0,44
3	43	82	10	0,33	15	1,05
4	46	86	9	0,67	14	3,30
5	48	64	11	0,66	14	9,57
6	60	60	9	0,26	12	11,47
7	118	61	12	0,89	17	4,99
8	158	81	12	1,30	15	10,61
9	200	102	17	1,01	16	15,85
10	322	336	15	1,93	17	19,66
11	404	204	13	1,76	18	17,62
12	608	306	16	2,23	18	29,65
13	742	373	16	2,61	16	27,66
14	812	408	16	3,39	17	26,96
15	1466	735	14	3,38	16	53,24

Tabla 4.2. Tiempo de visualización de modelos OE y SP-Octrees.

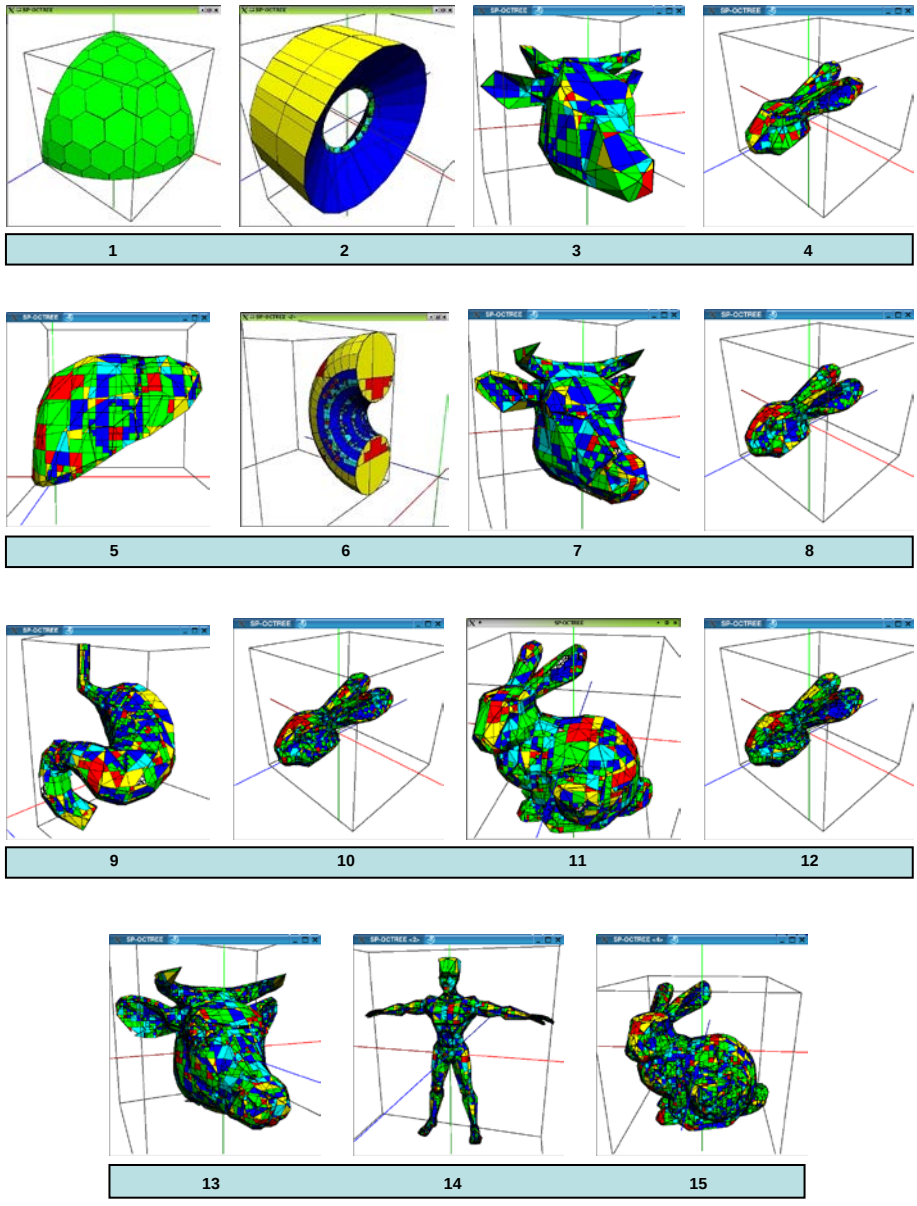


Figura 4.10. Modelos *SP-Octree* utilizados para evaluación de algoritmos.

La figura 4.11 muestra gráficamente la relación entre los dos algoritmos para la visualización de los modelos utilizados en la comparativa. Como podemos observar, en este caso el coste de tiempo necesario para SP-Octrees es mayor que para OE.

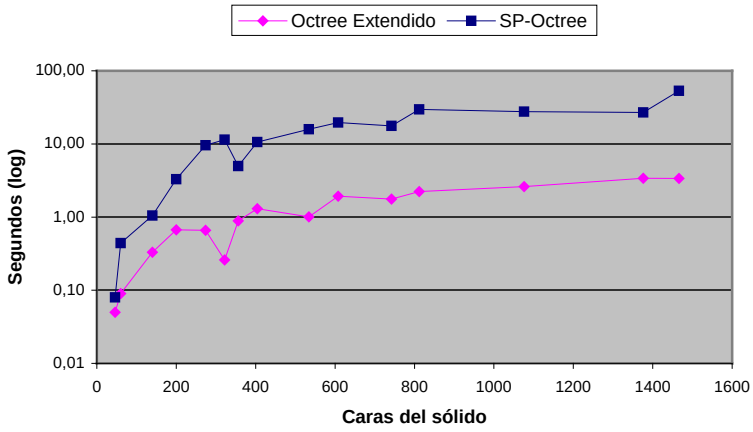


Figura 4.11. Tiempo para visualización (seg.) de modelos OE y SP-Octrees.

Al igual que ocurre en el algoritmo de clasificación de puntos, aunque el número de nodos totales y la profundidad del árbol en el modelo Octree Extendido sean mayores que en un SP-Octree, esta diferencia de tiempos en la visualización se debe a la mayor complejidad de los cálculos necesarios para procesar los nodos de un SP-Octree.

Debemos tener en cuenta que en los nodos de los modelos sólo se almacenan las referencias a los planos, de forma que no guardamos la geometría evaluada para minimizar el espacio necesario para el almacenamiento. Será en cada algoritmo que lo necesite cuando se evalúe dicha geometría para construir los polígonos frontera del sólido que pertenecen a cada uno de los nodos.

La figura 4.12 muestra la relación entre el número de nodos totales del árbol y el tiempo empleado para visualizar el modelo. En la figura 4.13 vemos la relación de ese tiempo con la profundidad del árbol.

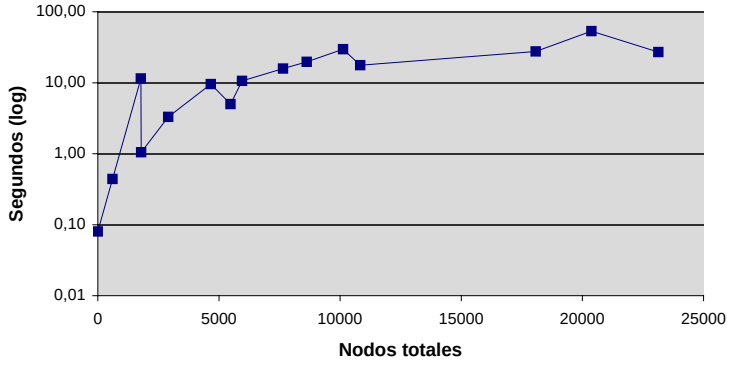


Figura 4.12. Tiempo para visualización de SP-Octrees según nodos del árbol.

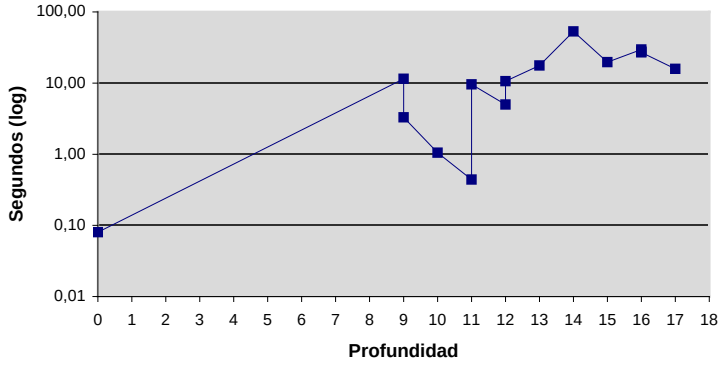


Figura 4.13. Tiempo para visualización según profundidad del árbol.

Para visualizar un nodo *cara* de un Octree Extendido, sólo se debe recortar el plano que aparece en el nodo con la caja englobante.

En el caso del nodo *arista*, se realiza el recorte de sus dos planos entre ellos (dependiendo de la configuración de la arista) y con la caja englobante. Tan solo el nodo vértice mantiene la misma complejidad.

Como hemos visto, en SP-Octree necesitamos realizar el recorte entre un mayor número de planos en cada nodo y, además, con los planos en los nodos ancestro por los que pasamos al descender en el árbol.

Sin embargo, el nuevo esquema propuesto presenta la ventaja de que el número de polígonos totales a visualizar una vez construidos es menor. Esto se debe a que, al ser inferior el número de niveles del árbol, los planos que aparecen en los nodos internos sufren menos divisiones que en SP-Octrees.

4.3.2. Visualización adaptativa

Como hemos visto en el apartado anterior, nuestro esquema permite visualizar el modelo por niveles, obteniendo representaciones visuales aproximadas al modelo original.

Esto nos permite también definir un nivel máximo a representar, no para todo el árbol, sino para ramas concretas del mismo. Haciendo esto en función de algún criterio establecido, podemos obtener una visualización adaptativa del modelo que permita representar con más detalle determinadas partes del mismo.

Dependiendo de la posición del observador o de la zona de interés establecida, para representar con más detalle parte del modelo debemos recorrer el árbol profundizando más en determinadas ramas del mismo.

En la figuras 4.14, para el caso de un *quadtree* de cuatro niveles, mostramos en color azul las ramas de los árboles que son recorridas para visualizar los nodos indicados en las imágenes superiores.

Aplicando el mismo mecanismo para los árboles octales de un SP-Octree podemos hacer que determinados nodos se visualicen con un nivel de profundidad distinto.

En la figura 4.15 vemos ese proceso en tres direcciones distintas: más detalle en los nodos superiores (arriba), más detalle en los nodos izquierdos (centro) y más detalle en los nodos cercanos al observador (abajo).

En cada caso se ha aumentado, de izquierda a derecha, el nivel máximo visualizado con menos detalle, y en función de este se ha ido aumentando a medida que nos acercamos a la zona con más detalle.

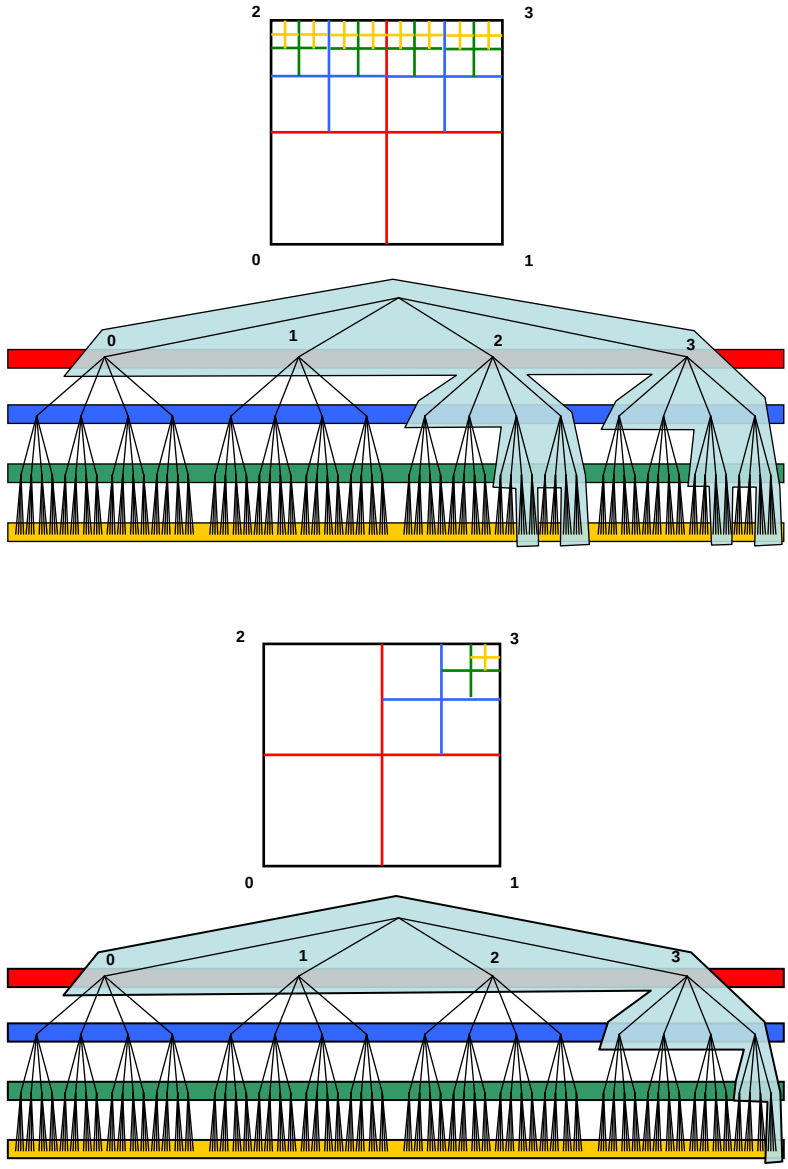


Figura 4.14. Visualización adaptativa para *Quadtrees.*

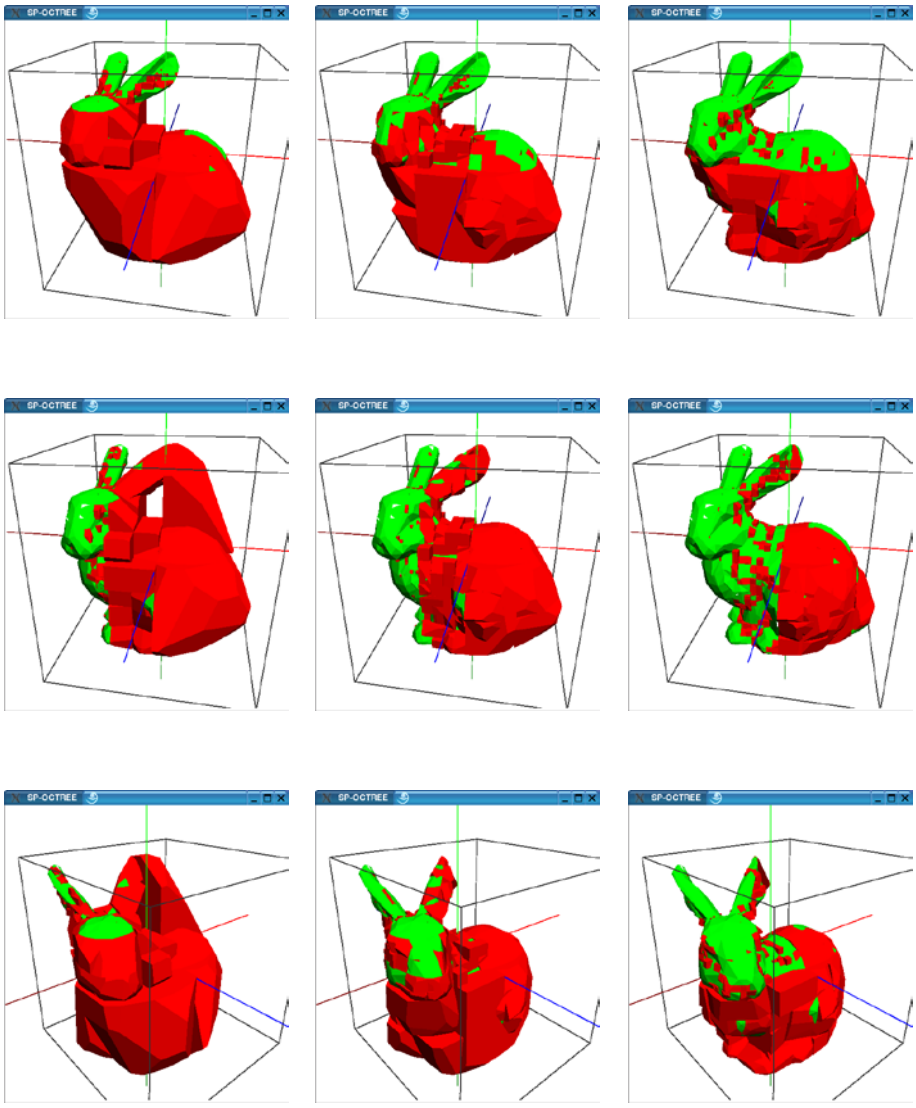


Figura 4.15. Visualización adaptativa de *SP-Octrees.*

4.4. Conversión desde/a otros esquemas

En los sistemas de modelado de sólidos y diseño por ordenador actuales es muy común utilizar más de un esquema de representación del modelo de forma que se use en cada caso el más apropiado para la operación realizada.

Pero para mantener la integridad entre los distintos esquemas es necesario definir mecanismos de conversión de una representación a otra para mantener la coherencia en la información almacenada.

Como cualquier otro esquema de representación, el método propuesto en este trabajo presenta características que lo hacen más apropiado para determinadas operaciones, pero menos interesante para otras. Por ello, puede ser útil utilizarlo en conjunción con algún otro método de representación que sea más adecuado en esos casos. A continuación describimos los algoritmos de conversión entre Octrees Extendidos, B-Rep y el modelo propuesto.

4.4.1. De/a B-Rep

El proceso de construcción de un modelo en la estructura propuesta a partir de la representación de un sólido poliédrico mediante las caras que forman su frontera ya ha sido descrito con detalle en el apartado 3.5 de esta memoria. Pasamos ahora a discutir algunos aspectos importantes del mismo, comparándolo con el proceso de construcción de Octrees Extendidos.

Como vimos dicho proceso es similar al de construcción de Octrees Extendidos con la salvedad del tratamiento realizado en los nodos internos y el criterio de clasificación de nodos.

En nuestro caso, el principal problema que se plantea es la complejidad de esa clasificación, ya que para cada nodo debemos comprobar si alguna de las caras que aparecen en él forma parte de la envolvente convexa de la parte de sólido que pertenece al nodo.

Además, debemos comprobar la configuración de todas las aristas que aparecen en el nodo para poder clasificar los nodos como *cóncavos y convexos.* Mientras que en Octrees Extendidos sólo es necesario comprobar la configuración de una arista en el caso de nodos *arista*, ahora el número de caras

que pueden aparecer en un nodo no está limitado, por lo que el número de aristas puede ser elevado, complicando así el proceso de clasificación.

Por último, debemos hacer notar que, como ocurre con los Octrees Extendidos, la inclusión del nodo *vértice* permite representar de forma exacta cualquier objeto poliédrico. Sin embargo, la variedad de configuraciones que pueden aparecer hace que este tipo de nodo sea el que más espacio necesite para su almacenamiento y el más complejo de procesar en todas las operaciones realizadas con el modelo.

4.4.1.1. Tiempo de construcción

En SP-Octrees el tipo de cada nodo no depende del número de vértices o aristas que aparecen en él como ocurre en los Octrees Extendidos, sino de la configuración que estas presentan. El proceso de clasificación necesita obtener la configuración de todas las aristas que aparecen en un nodo, por lo que el número de caras del poliedro a representar y la complejidad de las mismas influye directamente en el algoritmo de construcción.

Sólido			Octree Extendido		SP-Octree	
Modelo	**Planos**	**Vértices**	**Nivel**	**Construcción (seg.)**	**Nivel**	**Construcción (seg.)**
1	46	86	0	0,18	7	4,89
2	60	60	11	2,39	11	4,77
3	140	72	10	14,28	15	31,08
4	200	102	9	51,16	14	150,82
5	274	139	11	68,03	14	113,08
6	322	336	9	108,94	12	138,27
7	356	180	12	130,05	17	206,24
8	404	204	12	293,80	15	305,38
9	534	269	17	277,16	16	290,92
10	608	306	15	836,11	17	2402,21
11	742	373	13	1456,70	18	3224,26
12	812	408	16	1544,59	18	4385,00
13	1076	540	16	2028,31	16	1926,82
14	1376	714	16	3759,57	17	3504,47
15	1466	735	14	3613,59	16	4430,20

Tabla 4.3. Tiempo de construcción de modelos *OE* y *SP-Octrees.*

En la tabla 4.3 mostramos el tiempo, en segundos, utilizado por el algoritmo de construcción descrito para los modelos de los sólidos de la figura 4.9. También se incluye el tiempo del algoritmo de construcción del modelo OE.

La figura 4.16 muestra la relación entre esos tiempos, donde se ve claramente que, a pesar de ser más costoso el tiempo de clasificación de nodos en el esquema SP-Octree, la menor profundidad del árbol y el menor número de nodos totales hace que el proceso de construcción sea, generalmente, más rápido.

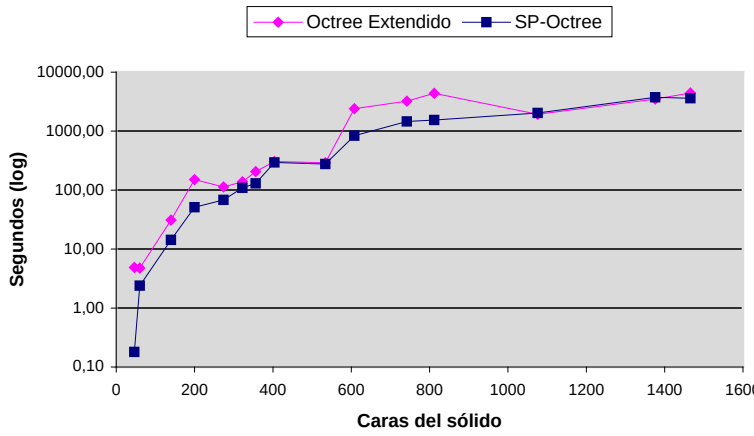


Figura 4.16. Tiempo de construcción de modelos *OE* y *SP-Octrees.*

4.4.1.2. Reconstrucción de la frontera

Al no tener almacenada explícitamente la geometría en cada uno de los nodos sino la referencia a los planos que soportan dicha geometría, para reconstruir la frontera del sólido representado debemos avaluar dicha geometría en función de la información almacenada en cada nodo.

Mediante el proceso descrito para la visualización de los objetos representados mediante el esquema SP-Octree podemos fácilmente reconstruir sin pérdida de información la frontera del sólido siguiendo los mismos pasos descritos en ese apartado.

En este caso, cuando obtenemos el polígono asociado a cada plano dentro de cada nodo, lo insertamos en una lista de polígonos asociados a ese plano. De esta forma, cuando un plano es compartido por varios nodos vecinos, la cara frontera del sólido asociada al mismo se encuentra dividida en varios polígonos que se incluyen en la lista asociada al plano.

Una vez construidas esas listas de polígonos, necesitamos realizar un postproceso para unir los polígonos coplanarios asociados a cada plano para reducir el número de caras poligonales utilizadas para representar la frontera del sólido.

A partir de esos polígonos finales, podemos obtener la lista de vértices y aristas del modelo para construir una representación B-Rep completa.

4.4.2. De/a Octrees Extendidos

La equivalencia directa existente entre los tipos de nodos utilizados en los Octrees Extendidos y los SP-Octrees permite realizar la conversión entre estos dos esquemas de forma sencilla.

En una primera fase, para pasar de un modelo OE a SP-Octree basta con recorrer el árbol, transformando cada uno de los nodos en su correspondiente equivalencia en el nuevo esquema mediante las siguientes reglas:

- Los nodos *blancos* y *negros* siguen quedando igual.
- Cada nodo *cara* se transforma en un nodo *convexo* con la referencia al plano correspondiente.
- Cada nodo *arista* se transforma en un nodo *convexo* o *cóncavo* en función de la configuración de la arista. En cualquier caso, se almacena sólo la referencia a los dos planos que comparten la arista.
- Los nodos *vértice* podrán convertirse en *vértice, convexo o* cóncavo, dependiendo de la configuración de las aristas que convergen en el vértice: si todas ellas son cóncavas, el nodo creado será *cóncavo;* si son convexas, el nodo será *convexo;* y si existen de los dos tipos el nodo será *vértice.*
- Los nodos *grises* se mantienen exactamente igual.

Pero con esta primera fase los planos pertenecientes a la envolvente convexa de cada nodo interno creado se encuentran repetidos en sus nodos hijo y no hay información alguna en los nodos internos. Debemos por tanto, en una segunda fase del algoritmo, conseguir eliminar esa redundancia y subir a los nodos internos aquellos planos que pertenezcan a la envolvente convexa correspondiente.

Por tanto, una vez realizada la conversión, volvemos a recorrer la estructura creada, desde los niveles más profundos del árbol hacia arriba, realizando una *compactación* de los nodos del mismo.

Para cada nodo *gris* se estudian sus ocho nodos hijos comprobando si alguno de los planos que aparecen en los mismos forma parte de la envolvente convexa del sólido representado por todos esos nodos. Si es así, lo subimos al nodo padre y se elimina de los nodos hijo donde aparezca. En ese caso, es posible que el tipo de cada nodo pueda cambiar, debiendo actualizarlo si es preciso. Este proceso se repite de forma recursiva hasta llevarlo al máximo nivel posible.

Además, en los nodos *grises* con nodos hijo *cóncavos*, puede ocurrir que al subir a los ancestros algunos de los planos, en los nodos hijos resultantes sólo queden los planos que forman concavidad, por lo que se podrán compactar en un nodo terminal *cóncavo*.

Si al compactar un nodo, sus ocho hijos terminan sin ningún plano significará que el nodo padre se ha convertido en nodo terminal, por lo que habrá que eliminarlos de la estructura.

La figura 4.17 muestra un ejemplo de este proceso. Para convertir el Octree Extendido de la izquierda en su correspondiente SP-Octree de la derecha, los pasos realizados son los siguientes:

a) Convertimos directamente los tipos de nodos OE a tipos SP-Octree:

   En el ejemplo los nodos *vértice* convexos (verdes) se transforman a nodos *convexos* verdes); los nodos *vértices* con concavidades y convexidades se transforman a nodos *vértice* (celeste); los nodos *arista* convexas (amarillo) se transforman a nodos *convexos;* y los nodos *arista* cóncavos se transforman a nodos *cóncavos* (azul).

b) Subimos planos del nivel 2 por pertenecer a la envolvente convexa del nivel 1. Los nodos *vértice* se convierten en *cóncavos*.

c) Subimos planos del nivel 1 por pertenecer a envolvente del nivel 0.

d) Los nodos terminales vecinos *cóncavos y convexos* del nivel 2 comparten los mismos planos, por lo que se compactan en un nodo *cóncavo* del nivel 1.

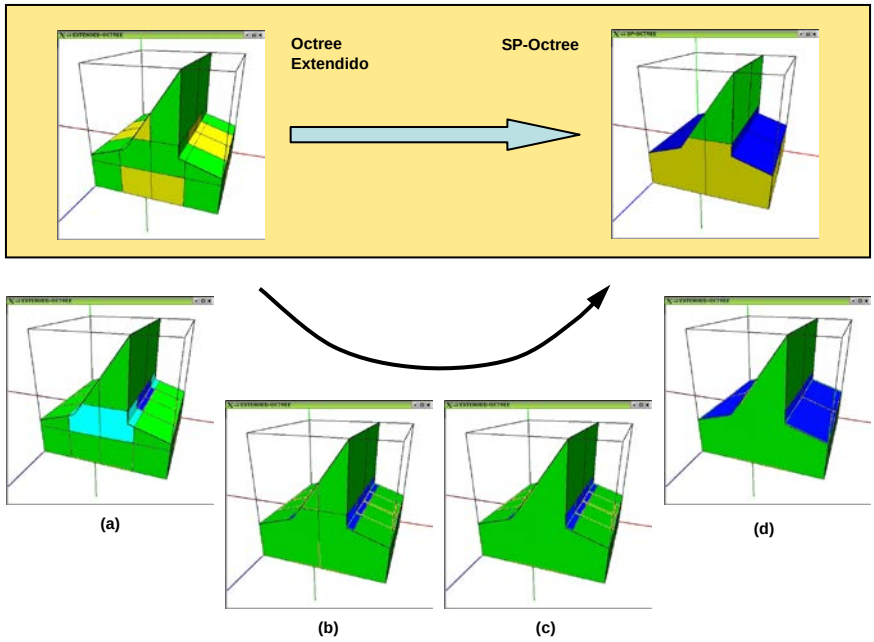


Figura 4.17. Conversión de *Octree Extendido* a *SP-Octree.*

La conversión en sentido opuesto, de SP-Octree a Octree Extendido, utiliza el propio algoritmo de construcción de OE a partir de la información geométrica reconstruida en los nodos terminales del SP-Octree.

Comenzamos el proceso recorriendo el árbol en preorden, empujando los planos que van apareciendo en los nodos internos hacia abajo insertándolos en los nodos hijos. Cuando alcancemos un nodo terminal, se reconstruye la frontera del sólido representado en el nodo y se aplica el algoritmo de construcción de Octrees Extendidos.

4.5. Conclusiones

En este capítulo se han descrito los algoritmos básicos necesarios para poder trabajar sobre modelos creados en el esquema propuesto. Las operaciones implementadas son algunas de las más interesantes y necesarias al trabajar con objetos sólidos.

Como en cualquier esquema jerárquico, todos los algoritmos propuestos se basan en realizar un determinado recorrido sobre el árbol de la estructura, operando sobre cada uno de los nodos por los que pasamos.

Basándonos en los métodos existentes para clasificación de puntos en modelos sólidos, se ha descrito el algoritmo utilizado en el esquema SP-Octree para comprobar la pertenencia al sólido de un punto, a través de la clasificación del mismo respecto a los nodos del modelo. Se ha realizado además una comparación con uno de los métodos existentes para realizar la clasificación sobre malla de triángulos [Feit98], así como con un algoritmo similar para Octrees Extendidos.

Se ha descrito el proceso de visualización del modelo, estudiando su complejidad y discutiendo la posibilidad de utilizar el esquema para realizar visualizaciones del modelo por niveles o de forma adaptativa.

Por último, se han presentado los algoritmos que permiten convertir un modelo en el esquema SP-Octree a dos de los esquemas utilizados para modelado de sólidos, y viceversa.

CAPÍTULO

5

Operaciones Booleanas

5

Unas de las operaciones más importantes en modelado de sólidos son las operaciones booleanas, que permiten crear nuevos objetos a partir de otros ya existentes. En este capítulo presentamos la realización de dichas operaciones entre modelos representados con el esquema propuesto, describiendo los algoritmos geométricos utilizados para realizarlas.

5.1. Introducción

Las operaciones booelanas (intersección, unión y diferencia) son un método muy potente y a la vez sencillo para poder construir modelos más complejos a partir de otros más simples. Implementar estas operaciones sobre cualquier esquema de representación ofrece un mecanismo muy útil de creación de modelos que por otros mecanismos serían complicados de construir.

Sin embargo, debemos tener en cuenta que al realizar estas operaciones sobre modelos de sólidos puede ocurrir que el resultado no sea un sólido válido. Para evitarlo debemos utilizar *operaciones booleanas regularizadas* (apartado 2.2.4).

Una de las principales ventajas que presenta el esquema de representación Octree clásico es que las operaciones booleanas son sencillas y rápidas, con una complejidad lineal del orden del número de nodos del árbol, *O(número de nodos)*.

Por el contrario, realizar esas operaciones booleanas entre modelos B-Rep implica operar todas las caras frontera de un objeto con todas las del otro, por lo que es un proceso muy costoso, de orden $O(n^2)$ siendo n el número de caras.

5.2. Operaciones booleanas con modelos jerárquicos

Las operaciones booelanas con Octrees clásicos son muy simples ya que no necesitan ningún cálculo geométrico. Se basan en recorrer los árboles de los dos modelos simultáneamente e ir operando los nodos que ocupan la misma posición en los dos árboles [Meag80]. El resultado depende del tipo de los nodos y de la operación a realizar.

Debemos tener en cuenta para estos algoritmos que los dos modelos deben tener cajas englobantes idénticas en tamaño y colocadas en la misma situación.

Para obtener el modelo *complementario* de un Octree basta con recorrer el árbol y cambiar los nodos *Blancos* por *Negros* y viceversa. Para la *intersección*, recorremos los árboles operando los nodos homólogos (mismo tamaño y situación) creando en cada caso un nodo cuyo tipo viene dado por la tabla 2.1.

A ∩ B		Octree A		
		Blanco	Negro	Gris
Octree B	Blanco	Blanco	Blanco	Blanco
	Negro	Blanco	Negro	*Copia* A
	Gris	Blanco	*Copia* B	*Recursión*

Tabla 5.1. Intersección entre dos nodos en modelos Octrees.

El cálculo de la unión y la diferencia puede realizarse basándonos en las dos operaciones anteriores y las propiedades de las operaciones booleanas:

$$A - B = A \cap \overline{B} \tag{5.1}$$

$$A \cup B = \overline{\overline{A} \cap \overline{B}} \tag{5.2}$$

o utilizando tablas similares a la anterior que definan el resultado de la operación en cada caso (tabla 2.2).

Estas operaciones son lineales con una complejidad proporcional al número de nodos de los árboles, ya que se basa en un recorrido de los árboles a operar. Sin embargo, debemos tener en cuenta que en el proceso recursivo puede ocurrir

que obtengamos alguna vez 8 hijos terminales iguales, por lo que se debe realizar un proceso de compactación del árbol.

A ∪ B		Octree A		
		Blanco	Negro	Gris
Octree B	Blanco	Blanco	Negro	*Copia A*
	Negro	Negro	Negro	Negro
	Gris	*Copia B*	Negro	*Recursión*

A – B		Octree A		
		Blanco	Negro	Gris
Octree B	Blanco	Blanco	Negro	*Copia A*
	Negro	Blanco	Blanco	Blanco
	Gris	Blanco	Gris	*Recursión*

Tabla 5.2. Unión y diferencia de nodos en modelos Octrees.

Las operaciones booleanas en las extensiones propuestas a los Octrees clásicos, entre ellas los Octrees Extendidos, se basan en el mismo proceso recursivo que recorre los árboles de forma simultánea para ir operando los nodos homólogos. Los nodos clásicos de estos Octrees Extendidos se siguen operando de la misma forma, y por tanto sólo es necesario añadir el tratamiento de las operaciones para los nuevos nodos que aparecen en los esquemas.

Pero estas operaciones en los Octrees Extendidos pierden la simplicidad que presentan en los Octrees clásicos ya que es necesario realizar operaciones geométricas entre los distintos planos que aparecen en cada nodo [Ayal88, Brun90a, Ayal91].

Para obtener el complemento de un Octree Extendido, debemos recorrer el árbol cambiando el tipo y la información almacenada en cada nodo de la siguiente forma [Brun85]:

- los nodos *blancos se convierten en* negros, y viceversa,
- los nodos *grises* se mantienen iguales,
- las referencias a los planos en nodos *cara, arista y vértice* se mantienen iguales,
- la configuración de las aristas de los nodos *arista y vértice* cambia de cóncava a convexa y viceversa, y
- se invierte la orientación de los planos en la lista auxiliar.

El algoritmo de intersección se basa en el recorrido de los dos árboles operando los nodos homólogos igual que en Octrees clásicos, con la salvedad de que ahora el número de combinaciones posibles es mayor para operar los nuevos nodos terminales permitidos [Nava86, Brun90a, Ayal91].

La tabla 2.3 resume las distintas posibilidades con las que nos podemos encontrar al operar dos nodos del mismo tamaño y localización en los modelos OE a intersecar.

A ∩ B		Octree Extendido A					
		Blanco	Negro	Cara	Arista	Vértice	Gris
Octree Extendido B	Blanco	B	B	B	B	B	B
	Negro	B	N	*Copia A*	*Copia A*	*Copia A*	*Copia A*
	Cara	B	*Copia B*	B, C, A, G	B, C, A, V, G	B, C, A, V, G	*Recursión*
	Arista	B	*Copia B*	B, C, A, V, G	B, A, V, G	B, A, V, G	*Recursión*
	Vértice	B	*Copia B*	B, C, A, V, G	B, A, V, G	B, V, G	*Recursión*
	Gris	B	*Copia B*	*Recursión*	*Recursión*	*Recursión*	*Recursión*

Tabla 5.3. Intersección de nodos en modelos Octrees Extendidos.

Aunque el resto de operaciones booleanas sobre Octrees Extendidos (unión y diferencia) pueden realizarse mediante algoritmos similares al descrito en la intersección, para evitar problemas de robustez [Hoff89b, Brun90a] se definen a partir de las dos operaciones descritas y las propiedades definidas por las ecuaciones 5.1 y 5.2.

5.3. Operaciones booleanas con SP-Octrees

Las operaciones booleanas con el modelo propuesto se basan en el mismo algoritmo que el descrito para Octrees clásicos y Octrees Extendidos, añadiendo el tratamiento de los nuevos nodos terminales ahora utilizados.

Además, en nuestro caso debemos tener en cuenta el procesamiento de los nodos *grises* ya que, al almacenar información de parte de la frontera del

modelo, no solo debemos procesar recursivamente sus hijos, sino que tendremos que operar también con dicha información.

Al igual que ocurre con los Octrees Extendidos [Brun90a, Ayal91], las operaciones booelanas con nuestro modelo presenta las mismas ventajas sobre los modelos de fronteras al conocer el tipo de geometría que aparece en cada nodo. De esta forma, el número de posibles configuraciones geométricas con que nos encontramos al realizar la operación entre nodos estará restringido por el tipo de nodos utilizados, y, en general, su procesamiento será más sencillo que al operar los dos modelos de fronteras.

Sin embargo, aunque el procesamiento del nodo *vértice* en el esquema propuesto es similar al de Octrees Extendidos, el número de planos que pueden aparecen en los nodos *cóncavos* y *convexos* no está limitado, por lo que los calculos necesarios para esos nodos son normalmente más complejos que para los nodos *cara* y *arista* de los Octrees Extendidos.

5.3.1. SP-Octree complementario

El cálculo del objeto complementario en un modelo SP-Octree se obtiene invirtiendo la orientación de los planos almacenados en la matriz auxiliar y recorriendo el árbol modificando los nodos siguiendo las siguientes reglas:

- Los nodos *blancos* se convierten en nodos *negros,* y viceversa.
- Los nodos *convexos* se convierten en *cóncavos* y viceversa.
- Los nodos *vértice* siguen siendo del mismo tipo, con la salvedad de que debemos invertir la configuración de las aristas que aparecen en el nodo.
- Cuando procesamos un nodo *gris*, en el objeto complementario los planos convexos que aparecían en él pasan a formar concavidades con los planos que aparecen en los hijos.

  En este caso el resultado será un nodo *gris* sin planos ya que estos son transmitidos a los nodos hijos donde al aplicarles las reglas de cambio anteriores debemos tener en cuenta esos nuevos planos para aplicarles el proceso descrito de forma recursiva.

En este proceso debemos tener en cuenta que no todos los planos deben transmitirse a todos los hijos, sino que sólo debemos transmitir a cada hijo aquellos planos que corten ese nodo y que pertenezcan a la frontera del sólido representado en el mismo.

Si el nodo hijo es un nodo *gris*, debemos seguir transmitiendo hasta los nodos terminales los planos acumulados de forma recursiva.

Los planos del nodo *gris* no se transmitirán a hijos *blancos* ni *negros* que estén totalmente dentro del sólido (no están recortados por los planos de sus ancestros), convirtiéndose en *negros* y *blancos* respectivamente.

Si un nodo hijo es *negro* pero está recortado por planos existentes en los ancestros, al recibir los planos de su nodo padre para crear el complementario se tratará como nodo *convexo.*

Un nodo hijo *convexo*, al recibir los planos de su nodo padre seguirá siendo *convexo.*

Por último, al empujar los planos de un nodo *gris* hacia un hijo *cóncavo o vértice*, el nodo resultado será también un nodo *gris, en* el que debemos reconstruir la geometría perteneciente al nuevo nodo para aplicar después el algoritmo de clasificación utilizado en la construcción del modelo.

En la figura 5.1 podemos ver un ejemplo de la construcción del SP-Octree complementario en un nodo *gris* para ilustrar el algoritmo propuesto. Se ha utilizado un ejemplo en dos dimensiones para facilitar la interpretación del proceso.4

Como podemos observar, para los nodos terminales el proceso es similar al utilizado en Octrees Extendidos. Sin embargo, al aparecer información adicional en los nodos *grises*, el proceso de estos nodos es más complejo.

Además, el número de niveles del SP-Octree complementario puede ser mayor que el del SP-Octree original ya que al enviar los planos en nodos internos hacia hijos *cóncavos* o *vértices* aparecen nuevos nodos *grises*.

Los algoritmos 5.1 y 5.2 muestran los pasos descritos para obtener el complementario de un SP-Octree.

```
FUNCTION ComplementoSPO (SPOctree)
{
  Invertir coeficientes de planos en matriz auxiliar;
  ComplementaNodo (NodoRaiz(SPOctree));
}
```

Algoritmo 5.1. Complemento de un SP-Octree.

```
FUNCTION ComplementaNodo (Nodo)
{
  if (Tipo(Nodo) == GRIS)
  {
    Transmitir planos de Nodo a sus nodos hijo;
    for (i=0; i<8; i++)
    {
      if ((TipoNodo(Nodo.Hijo[i]) == NEGRO) &&
          (Nodo.Hijo[i] es recortado por planos del padre))
        Tipo(Nodo.Hijo[i]) = CONVEXO;

      if (TipoNodo(Nodo.Hijo[i]) == CONCAVO o VERTICE)
      {
        Tipo(Nodo.Hijo[i]) = GRIS;
        ListaPoligonos = Reconstruir geometría en el nodo;
        Construir_SPOctree(Nodo.Hijo[i], ListaPoligonos);
      }
      else
        ComplementaNodo (Nodo.Hijo[i]);
    }
  }
  else
    if (Tipo(Nodo) == BLANCO)
      Tipo(Nodo)= NEGRO;
    else
      if (Tipo(Nodo) == NEGRO)
        Tipo(Nodo) = BLANCO;
      else
        if (Tipo(Nodo) == CONVEXO)
          Tipo(Nodo) = CONCAVO;
        else
          if (Nodo es CONCAVO)
            Tipo(Nodo) = CONVEXO;
          else
            if (Nodo es VERTICE)
              Invertir Configuración Aristas;
}
```

Algoritmo 5.2. Complemento de un nodo en SP-Octree.

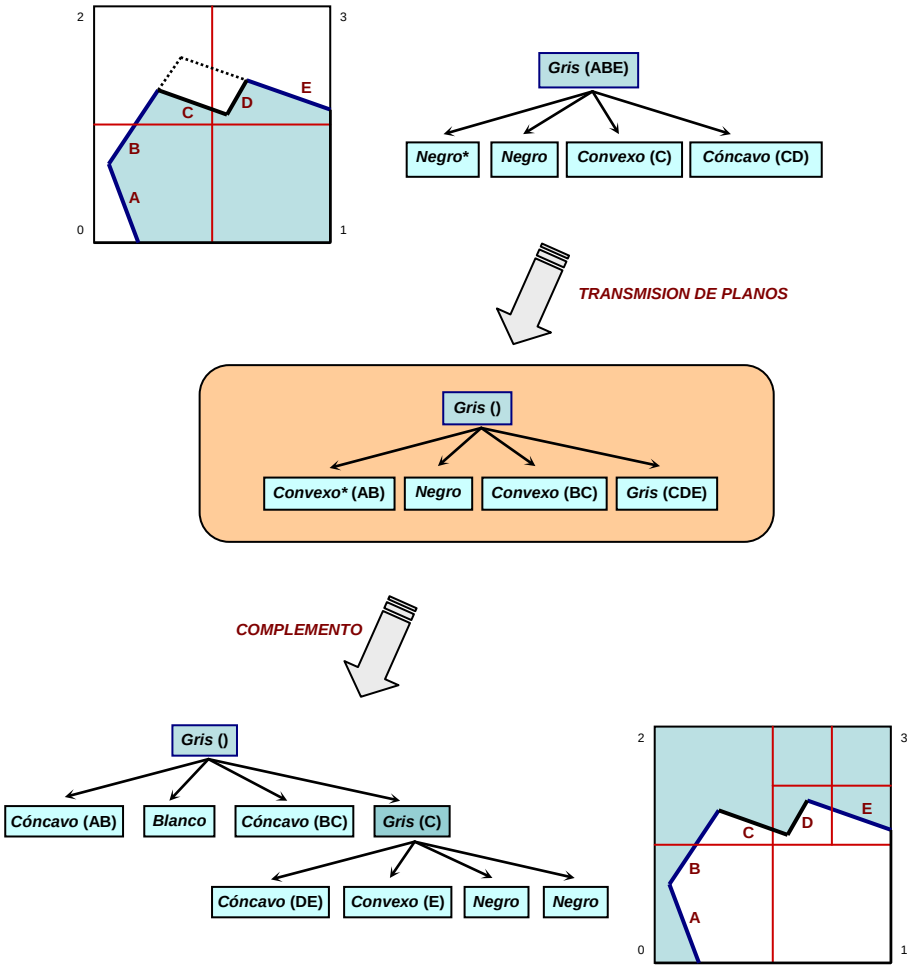


Figura 5.1. Complemento de nodo *gris* en SP-Octrees.

Un aspecto importante que debemos tener en cuenta a la hora de definir el modelo complementario es que al invertir los planos obtendríamos un sólido *infinito*. Para evitar este problema debemos delimitar el sólido representado, utilizando para ello la propia caja envolvente del SP-Octree original como límite.

5.3.2. Intersección entre SP-Octrees

El algoritmo para realizar la intersección entre dos modelos SP-Octree, consiste en recorrer sumultaneamente los dos árboles a intersecar, operando cada par de nodos homólogos dependiendo de la configuración que se presente.

Este proceso, al igual que en los Octrees Clásicos, necesita que ambos modelos tengan la misma caja englobante para que al procesar sus árboles podamos operar con nodos homólogos en cuanto a tamaño y posición.

El primer paso en el algoritmo será unir las listas auxiliares de planos asociados a cada modelo SP-Octree. Debemos tener en cuenta que si un plano aparece en ambos modelos, debemos insertarlo en la lista resultante una única vez para evitar redundancias.

Depués, recorrermos simultáneamente los dos árboles, de forma que cada par de nodos homólogos generarán como resultado uno o más nodos, dependiendo de su tipo, siguiendo las siguientes reglas:

- Si uno de los nodos es *BLANCO*, la intersección será un nodo *BLANCO*.
- Si uno de ellos es *NEGRO*, la intersección será la copia del otro nodo (o todo el subárbol si se trata de un nodo *gris)*.
- Si los dos nodos son *CONVEXOS*, el resultado podrá ser *BLANCO* o *CONVEXO*, dependiendo de si los sólidos representados en cada uno de los nodos intersecan entre si. Si hay intersección el nodo resultante será *CONVEXO*. Si no hay intersección, el resultado será un nodo *BLANCO*.

  Para comprobar dicha intersección obtenemos la frontera del sólido representado en cada uno de los nodos mediante el algoritmo descrito en el capítulo anterior. Una vez obtenidos los polígonos que las componen calculamos la intersección entre los polígonos de un nodo y los del otro.

  Si no hay intersección y los polígonos de un nodo son todos exteriores al sólido representado en el otro nodo, el resultado es un nodo *blanco*.

  Si no hay intersección entre ellos y todos los polígonos de uno de los nodos son interiores al sólido representado en el otro, el resultado será un nodo *convexo* con la referencia a los planos que soportan esos polígonos interiores.

Si alguno de los polígonos de un nodo interseca con los del otro, el resultado también será un nodo *convexo* con la referencia a los planos que soporten los polígonos de cada uno de los nodos que son interiores, total o parcialmente, al sólido representado en el otro nodo (forman parte de la frontera de la intersección de los dos sólidos convexos que estamos operando).

- Si los dos nodos son *CONCAVOS*, el resultado será *BLANCO* si los sólidos representados en cada nodo no intersecan. Si el sólido de un nodo es totalmente interior al otro, el resultado será un nodo *CONCAVO*.

  Si los dos sólidos intersecan parcialmente, el resultado podrá ser *CONVEXO*, *VERTICE* o *GRIS*, dependiendo de la situación de las concavidades de un nodo con respecto al otro.

  Para comprobar la intersección entre los sólidos representados en cada uno de los nodos, actuamos de la misma forma que en el caso anterior, reconstruyendo los polígonos frontera en cada uno de los nodos y realizando la intersección entre ellos teniendo en cuenta que pertenecen a nodos *cóncavos.*

- Si uno de los nodos es *CONVEXO*, calculamos si sus planos formarán parte de la envolvente convexa del resultado obteniendo su intersección con los planos del otro nodo. De nuevo, el nodo resultante es clasificado en función de la existencia o no de intersecciones entre dichos planos y de la situación del sólido representado en el otro nodo respecto a los planos del nodo *convexo*.

- Si uno de ellos es *CONCAVO*, el proceso es similar al punto anterior.

- Si alguno de los nodos es *VERTICE*, el resultado dependerá de dónde quede el vértice del nodo respecto al otro nodo.

  Si la intersección entre todos los planos es nula, y el vértice es interior al otro nodo, tenemos como resultado un nodo *VERTICE*. Si no hay intersección y el vértice es exterior al otro nodo, el resultado será *BLANCO*.

  Si al realizar la operación entre los distintos planos sigue quedando un único vértice, el resultado podrá ser un nodo *VERTICE* o *GRIS*, dependiendo de la configuración de las aristas resultantes.

  En otro caso, si tras operar los planos aparece más de un vértice, el resultado podrá ser *GRIS*, *CONVEXO* o *CONCAVO*, según el resultado de intersecar los planos entre si.

- Si uno de los nodos es *GRIS*, debemos operar el otro nodo con los planos existentes en el nodo *gris*, como si este se tratara de un nodo *convexo,* y propagar el resultado sobre sus hijos para recortar o eliminar planos en función de la nueva envolvente convexa obtenida en el padre resultante.
- Si los dos nodos son *GRISES*, deberemos calcular la intersección de los planos existentes en cada uno, como si se trataran de dos nodos *CONVEXOS.* Si el resultado de dicha intersección es *convexa*, el nodo resultante será *GRIS* , y debemos procesar recursivamente los hijos, propagando la información de los planos resultantes en el nodo padre obtenido para eliminar en los nodos resultantes de operar los nodos hijos de cada modelo, los planos que queden fuera de la nueva envolvente creada en el nodo *gris* obtenido.

Como comentamos al comienzo del apartado y podemos apreciar en el algoritmo descrito, el número de casos posibles con los que nos encontramos aumenta respecto a los que aparecen en la intersección de Octrees Extendidos debido a la mayor complejidad de los nodos *cóncavos* y *convexos* y del tratamiento de los nodos *grises.* La tabla 2.4 muestra un resumen de las distintas posibilidades que nos podemos encontrar y los resultados que se pueden obtener en cada una de ellas.

A ∩ B		SP-Octree A					
		Blanco	Negro	Convexo	Cóncavo	Vértice	Gris
SP-Octree B	Blanco	B	B	B	B	B	B
	Negro	B	N	*Copia A*	*Copia A*	*Copia A*	*Copia A*
	Convexo	B	*Copia B*	B, CX	B, CX, CC, V, G	B, CX, V, G	*Recursión*
	Cóncavo	B	*Copia B*	B, CX, CC, V, G	B, CX, CC, V, G	B, CX, CC, V, G	*Recursión*
	Vértice	B	*Copia B*	B, CX, V, G	B, CX, CC, V, G	B, CX, CC, V, G	*Recursión*
	Gris	B	*Copia B*	*Recursión*	*Recursión*	*Recursión*	*Recursión*

Tabla 5.4. Intersección de nodos en modelos SP-Octrees.

Como vemos, para calcular qué planos de cada nodo operando debe quedar en el nodo resultante, en todos los casos debemos reconstruir la frontera del sólido representado en cada nodo para operar con los polígonos resultantes de igual forma que en el modelo de fronteras [Mänt86, Putn86, Gurs91]. Sin

embargo, en nuestra propuesta conocemos a priori la configuración geométrica de cada nodo, reduciendo así el número de combinaciones posibles.

En algunos casos, como ya hemos comentado, es necesrio propagar el resultado de la intersección entre nodos *grises* hacia sus hijos para eliminar de esos nodos los planos que puedan quedar fuera de la envolvente definida en el nodo padre. Este proceso puede realizarse dentro del mismo procesamiento de los nodos hijos, o mediante un recorrido del árbol resultante después de realizar la operación booleana para compactar dicho árbol, eliminando planos innecesarios o uniendo nodos hermanos que compartan la misma configuración convirtiendo su nodo padre en terminal.

Debemos notar que en alguno de estos casos podríamos obtener sólidos *no-variedad* (aparecen vértices o aristas sobre caras del objeto resultante), por lo que con la definición actual del modelo no podemos representarlo (apartado 3.8).

En las figuras 5.2 y 5.3 podemos ver algunos ejemplos de la operación intersección descrita. En cada caso podemos observar los operandos utilizados y el resultado obtenido.

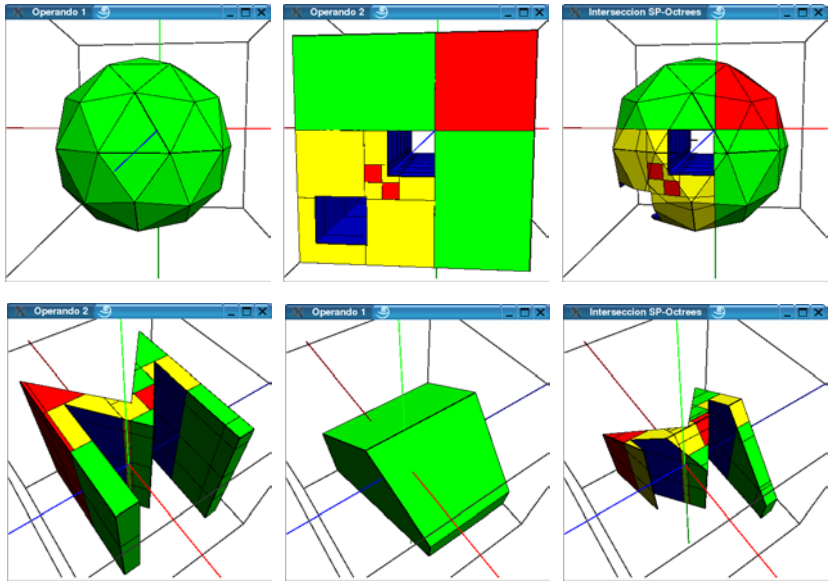


Figura 5.2. Intersección de modelos SP-Octree.

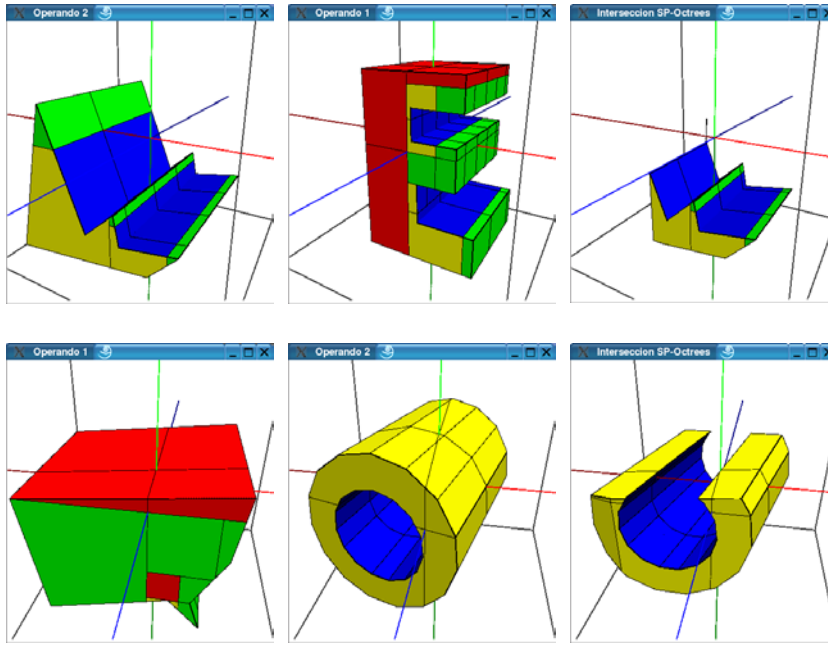


Figura 5.3. Intersección de modelos SP-Octree.

5.3.3. Unión y diferencia entre SP-Octrees

Podemos utilizar un algoritmo similar al descrito para la intersección para realizar las operaciones *unión y diferencia* entre modelos en el esquema propuesto, modificando las reglas de combinación de los distintos tipos de nodos utilizados como operandos en cada caso.

Sin embargo, como vimos en el apartado anterior para Octrees Extendidos, para evitar problemas de robustez en los algoritmos [Hoff89b], las operaciones de unión y diferencia entre modelos SP-Octrees pueden también realizarse en base a las operaciones de complemento e intersección descritas y las propiedades de las operaciones booleanas.

En la figura 5.4 podemos ver un ejemplo de la operación diferencia entre dos modelos SP-Octree utilizando la intersección entre un modelo y el complemento del otro.

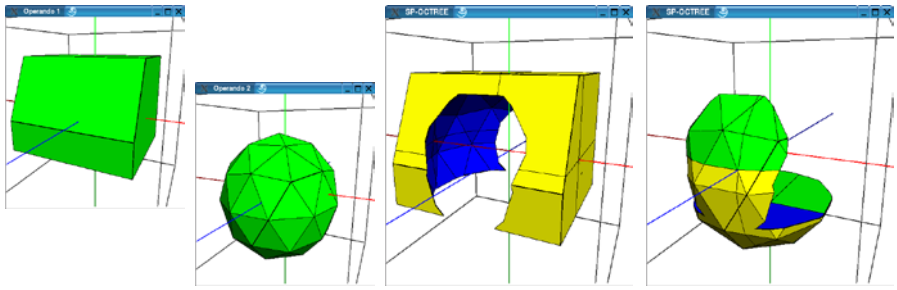


Figura 5.4. Diferencia de modelos SP-Octree.

5.4. Conclusiones

En este capítulo se han presentado y discutido los algoritmos necesarios para la realización de operaciones booleanas con el esquema propuesto.

Se han descrito los algoritmos para obtener el objeto complemento, así como la intersección de dos modelos SP-Octree en base a un recorrido simultáneo de los árboles que almacenan los modelos.

Para el resto de operaciones booleanas, aunque pueden utilizarse algoritmos similares al definido para la intersección, se definen en base a las dos operaciones anteriores: la intersección y el objeto complementario.

CAPÍTULO

6

Transmisión Progresiva con SP-Octrees

6

Con la utilización de sistemas distribuidos y el aumento de la complejidad de los modelos utilizados, en el campo del modelado de sólidos se hace necesario utilizar métodos de representación que permitan almacenar y transmitir de forma eficiente el modelo entre los distintos equipos que lo necesiten. En este capítulo presentamos la utilización del esquema propuesto para realizar la transmisión del modelo del sólido de forma progresiva, de manera que el receptor pueda trabajar con el modelo desde el comienzo de la transmisión.

6.1. Introducción

En los últimos años cada vez es más común encontrarnos, en cualquier ámbito de la informática, con sistemas distribuidos que comparten información para realizar su tarea. En estos sistemas nos encontramos con uno o varios *servidores* que almacenan la información con la que varios *clientes* trabajan, estando todos ellos interconectados por una red y, normalmente, ubicados en diferentes lugares, lo que hace necesario la transmisión de esa información entre ellos.

En la actualidad, centrándonos en el modelado de sólidos nos encontramos con la necesidad de representar objetos cada vez más complejos. Esto plantea problemas para encontrar esquemas de representación que puedan almacenar y tratar de forma eficiente ese volumen de información. Además, si el modelo va a tener que utilizarse en un sistema distribuido, como ocurre en el diseño cooperativo, el esquema utilizado para representarlo debe permitir su transmisión de forma eficiente.

Las mallas poligonales son una de las técnicas más utilizadas para almacenar modelos complejos 3D en multitud de aplicaciones [Kobb00a]. Las mallas

poligonales permiten aproximar cualquier superficie, de forma que el error cometido será menor cuanto más pequeños sean los polígonos utilizados.

El tipo de polígono más utilizado en esas mallas es el triángulo ya que presenta una geometría fácil de utilizar matemáticamente y cualquier otro polígono puede dividirse en triángulos sin dificultad.

Además, las librerías gráficas 3D más utilizadas (OpenGL [Neid93]), los lenguajes de descripción de mundos virtuales (VRML [Care97]) y las tarjetas gráficas para ordenadores actuales están optimizadas para el tratamiento de triángulos.

Sin embargo, el tamaño de los modelos representados mediante mallas de triángulos puede ser tan grande que su almacenamiento y transmisión en un sistema distribuido puede ser muy costosa.

Para resolver estos problemas en los últimos años se han propuesto métodos de simplificación de mallas triangulares que permiten reducir el número de triángulos y, por tanto, el tiempo necesario para procesar y transmitir el modelo [Pupp97, Roos97a, Roos97b].

Estos métodos permiten generar distintas versiones de un mismo modelo, con distinto nivel de detalle, que pueden ser usadas conjuntamente como un modelo *multirresolución* (figura 6.1) [Heck94, Garl99, Taub99].

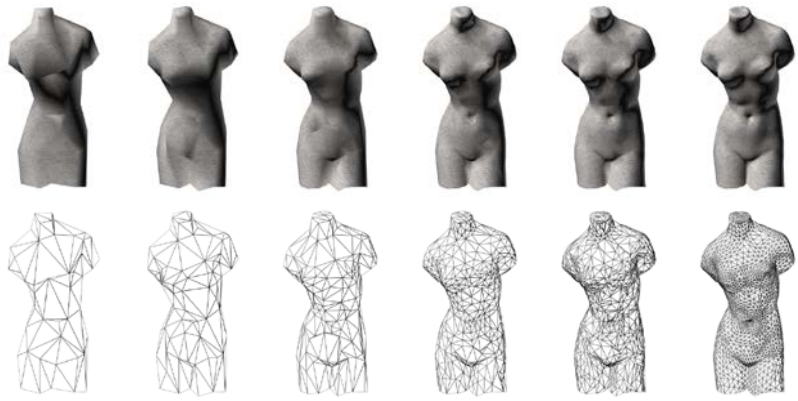


Figura 6.1. Varios niveles de detalle en un modelo *multirresolución.*

Podemos definir un modelo multirresolución como una representación de distintas aproximaciones del objeto real de forma que, en cualquier momento, se puede recuperar cualquiera de ellas [Garl99].

Dada una superficie frontera de un sólido, un modelo multirresolución de la misma es una secuencia de superficies, topológicamente equivalentes, que la aproximan con distinto nivel de detalle [Kobb00b]. En el proceso de visualización de un modelo multirresolución, esta representación permite utilizar las distintas aproximaciones dependiendo de la distancia del objeto al observador (ver figura 6.2), de forma que no sea fácil distinguir las diferencias existentes entre ellas.

Figura 6.2. Visualización de un modelo multirresolución según la distancia.

Se han propuesto múltiples estructuras multirresolución, desde un simple conjunto de distintas representaciones de un objeto a distinto nivel de detalle, a modelos más complejos que almacenan también las relaciones existentes entre niveles consecutivos de detalle.

Lo ideal es que un modelo pueda ser transmitido de forma inmediata con una primera aproximación al objeto, aunque sea de baja calidad, para que el receptor pueda comenzar a trabajar mientras se van enviando versiones mejoradas del modelo, hasta enviar la versión final.

Los modelos multirresolución pueden resolver el problema de esa transmisión enviando, de menor a mayor complejidad, los distintos niveles de detalle que mantienen del objeto. Sin embargo, la mayoría de estas representaciones se centran más en acelerar el proceso de visualización que en el proceso de transmisión del modelo, lo que puede provocar una transferencia redundante de información innecesaria.

Lo deseable sería enviar sólo las diferencias existentes entre niveles de detalle consecutivos, de forma comprimida, para que a partir de un nivel anterior se pueda reconstruir el siguiente.

Por otra parte, algunos de estos métodos permiten simplificar el modelo de forma continua, esto es, eliminando elementos geométricos uno a uno. Estos métodos pueden usarse en sentido inverso para enriquecer una malla simplificada, aumentando progresivamente el nivel de detalle [Hopp96, Paja00]. Esta estrategia es la usada habitualmente para realizar la transmisión progresiva de mallas de triángulos.

Una primera propuesta para la transmisión progresiva real de este tipo de modelos multirresolución son las *mallas progresivas* [Hopp96] con las que se envía en una primera fase una aproximación de baja calidad del modelo para después ir transmitiendo unos registros de modificación de la misma para ir refinando el modelo inicial.

De esta forma el receptor puede trabajar con el modelo tal y como se encuentre en un determinado instante mientras se continúa reconstruyendo a nivel local con los registros que vaya recibiendo.

No obstante, ninguna de estas técnicas está diseñada para trabajar con modelos sólidos, lo que puede provocar que aparezcan modelos no válidos en el proceso de simplificación.

6.2. Transmisión de modelos jerárquicos

Algunos de los esquemas de representación de sólidos y volúmenes se basan en la descomposición del espacio que ocupa el objeto representado y utilizan estructuras jerárquicas para almacenar el modelo.

Estos modelos jerárquicos utilizan normalmente estructuras en árbol para almacenar la información, lo que los hace idóneos para realizar una transmisión progresiva del modelo representado, enviando nivel a nivel los nodos del árbol [Vela02b].

Centrándonos en Octrees clásicos, si utilizamos distintos niveles de profundidad en el árbol, el modelo resultante en cada uno será una aproximación

del sólido original. Esta aproximación será mejor conforme vayamos descendiendo de nivel.

Con esta idea, podríamos realizar la transmisión del modelo de forma progresiva enviando nivel a nivel la información del árbol, con lo que el receptor, en cada nivel transmitido, tendría un modelo válido del sólido representado con el que trabajar (figura 6.3).

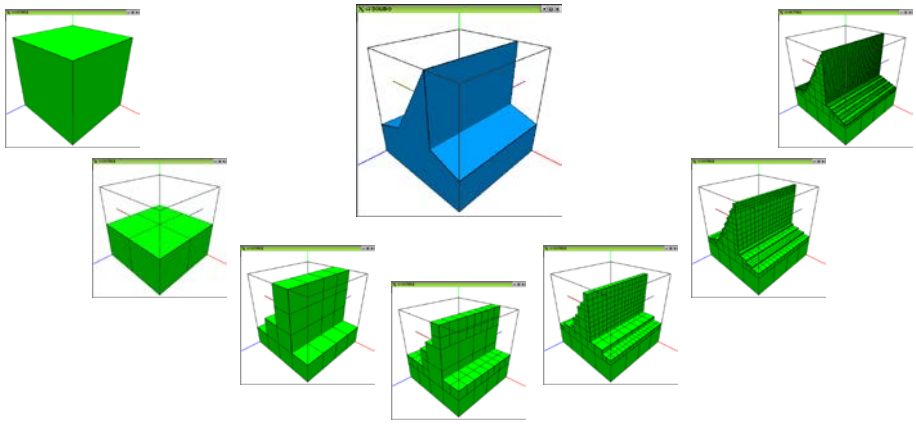


Figura 6.3. Transmisión progresiva de un Octree clásico.

En el esquema de representación propuesto, mantenemos información de parte de la frontera del sólido no solo en los nodos terminales, sino también en los nodos intermedios del árbol.

La información que guardamos en los nodos terminales forma parte de la frontera del sólido representado en esos nodos, como en los Octrees Extendidos. Además, en los nodos internos guardamos la información de la envolvente convexa de la parte del sólido representado en ese nodo.

De esta forma, si podamos todas las ramas del árbol a un mismo nivel, la representación resultante sigue siendo un modelo válido que representa una aproximación del sólido, la cual podemos ir mejorando a partir de la información almacenada en los niveles inferiores (como si se tratara de un proceso de modelado eliminando materia del sólido).

Al enviar de forma progresiva los distintos niveles del árbol, en la primera fase de la transmisión el receptor obtiene la envolvente convexa del sólido (formada por los planos que aparecen en el nodo raíz del árbol), y en las sucesivas transmisiones vamos enviando información de cómo ir *moldeando* el sólido mediante la eliminación de la parte de objeto que no pertenece al sólido en cada uno de los nodos del nivel inferior (figura 6.4).

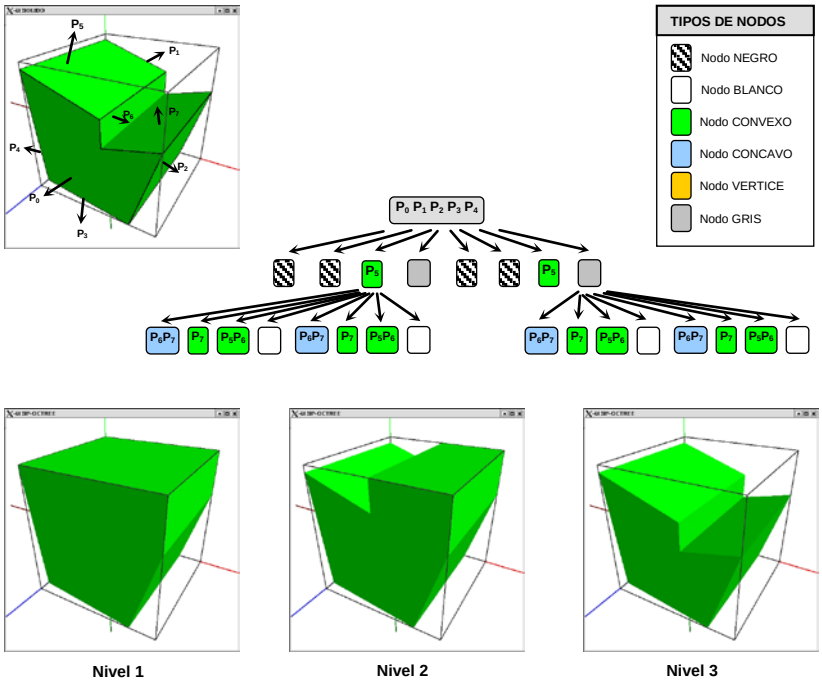


Figura 6.4. *SP-Octree* de un sólido y modelos obtenidos por niveles.

6.3. Transmisión progresiva de un *SP-Octree*

Como hemos visto en el apartado anterior, mantener información de la frontera del sólido representado en los nodos interiores del SP-Octree nos permite obtener representaciones aproximadas del objeto sin tener que descender hasta el máximo nivel del árbol.

Esto hace que, en un sistema cliente-servidor que tenga que compartir un objeto representado con esta estructura, el receptor pueda visualizar o trabajar con un modelo aproximado del sólido desde el inicio de la transmisión, sin necesidad de esperar la llegada de toda la representación.

Para realizar la transmisión de un modelo representado mediante un SP-Octree, debemos tener en cuenta que necesitamos transmitir tanto la información del árbol octal que mantiene la estructura, como la información geométrica que almacenamos en cada nodo de ese árbol.

6.3.1. Transmisión de información geométrica

En un SP-Octree puede aparecer información sobre parte de la frontera del sólido tanto en los nodos internos como en los terminales. Por tanto, a la hora de transmitir cada nivel del árbol necesitamos enviar la información geométrica de aquellos planos que aparecen en los nodos de ese nivel.

Aunque se podría transmitir la información de todos los planos que aparecen en el modelo antes de enviar la información del árbol, los planos no se utilizarán hasta que se transmita el nivel del árbol en el que aparece el nodo o nodos que contienen a dichos planos. Por ello, optamos por transmitir la información de los planos de forma progresiva según van apareciendo en la estructura.

Como vimos en el capítulo 3, esa información geométrica se mantiene almacenada en una estructura auxiliar, no dentro de cada nodo, para evitar repetir información en el modelo. En esa lista de planos se almacenan los coeficientes de la ecuación de cada uno de ellos. Al construir el modelo, la información geométrica de los planos frontera se va insertando en la lista auxiliar. Cada vez que aparece un nuevo plano a incluir en un nodo, si no estaba ya almacenado, se inserta en esa lista.

En los nodos del árbol lo que se almacena es un identificador que referencia la posición que ocupa el plano en esa estructura adicional.

Para transmitir la información geométrica de los planos que aparecen en un determinado nivel del árbol, se deberá recorrer los nodos de ese nivel enviando los planos que aparecen en ellos, siempre que no se hayan enviado ya. Para realizar esa comprobación, mantendremos unas marcas que indiquen cuándo un plano ya se ha enviado en niveles anteriores.

A continuación mostramos el pseudocódigo de la transmisión de la geometría de los planos de un nodo, teniendo en cuenta las marcas de planos ya enviados:

```
FUNCTION EnviarGeometria(Nodo, ListaEnviados)
{
   for (i=0; i<Nodo.NumPlanosNodo; i++)
      if (Nodo.P[i] ∉ ListaEnviados)
      {
         Enviar (CoeficientesPlano(Nodo.P[i]));
         Insertar Nodo.P[i] en ListaEnviados;
      }
}
```

Algoritmo 6.1. Envío de información geométrica de un nodo

La función *CoeficientesPlanos(i)* obtiene los coeficientes de la ecuación del plano que ocupa la posición i en la lista auxiliar de planos.

La variable *ListaEnviados* contendrá el índice de los planos que ya han sido enviados en niveles anteriores.

La recepción se realizaría de la misma forma, almacenando el receptor los planos recibidos en su lista auxiliar.

En la tabla 6.1 podemos ver la información geométrica almacenada para los ocho planos que aparecen en el árbol del modelo de la figura 6.4, mientras que en la tabla 6.2 mostramos los planos enviados al transmitir progresivamente cada uno de los niveles.

Como vemos en el árbol, en algunos nodos del tercer nivel aparece de nuevo la referencia al plano P_5, pero no debe transmitirse en ese tercer nivel ya que se envió al transmitir el segundo nivel.

En el envío del primer nivel debemos transmitir también la caja englobante del modelo completo. Con esta información y el nivel en el que nos encontremos, podemos obtener la información geométrica de la caja englobante de cada nodo.

PLANOS DEL MODELO	
Indice	**Geometría**
0	P_0 (a_0,b_0,c_0,d_0)
1	P_1 (a_1,b_1,c_1,d_1)
2	P_2 (a_2,b_2,c_2,d_2)
3	P_3 (a_3,b_3,c_3,d_3)
4	P_4 (a_4,b_4,c_4,d_4)
5	P_5 (a_5,b_5,c_5,d_5)
6	P_6 (a_6,b_6,c_6,d_6)
7	P_7 (a_7,b_7,c_7,d_7)

Tabla 6.1. Lista auxiliar de planos para el *SP-Octree* de la figura 6.4.

Nivel	Planos transmitidos
1	P_0 P_1 P_2 P_3 P_4
2	P_5
3	P_6 P_7

Tabla 6.2. Transmisión de información geométrica para la figura 6.4.

En el ejemplo de la figura 6.4 podemos ver cómo, al transmitir en el primer nivel los planos (P_0 P_1 P_2 P_3 P_4) junto con la caja englobante del sólido, tenemos una primera aproximación del sólido.

En los siguientes niveles vemos cómo los planos transmitidos, junto a los planos de los niveles anteriores y la caja englobante de cada nodo, van *eliminando* materia del sólido hasta obtener la representación exacta en el tercer nivel.

6.3.2. Transmisión de la estructura jerárquica

Además de la información geométrica debemos transmitir el árbol octal que soporta la estructura. Para ello, vamos recorriendo el árbol por niveles (según un recorrido *primero en anchura*) y enviando la información que aparece en cada nodo.

En cada nivel enviamos los hijos de los nodos grises del nivel superior. Para ello, al recorrer un nivel, mantenemos en una estructura *cola* los nodos grises que aparezcan, cuyos hijos deberán ser enviados en el siguiente nivel. Cuando completamos un nivel insertamos una marca de final de nivel en la cola.

Para transmitir un nuevo nivel, vamos sacando nodos de esa estructura cola y enviando sus hijos. Si alguno de esos nodos hijo es gris, lo insertamos de nuevo en la cola. Este proceso se repite hasta completar los nodos del nivel actual, momento en el que insertamos una nueva marca que indica el fin del siguiente nivel.

El algoritmo 6.2 muestra los pasos descritos para enviar los nodos de la estructura jerárquica, donde:

- *ColaGrises* mantiene los nodos grises cuyos hijos están pendientes de enviar en los siguientes niveles,

```
FUNCTION EnviarNodo (NodoRaiz)
{
  if (NodoRaiz=GRIS)
  {
    Insertar (ColaGrises, NodoRaiz);
    Insertar (ColaGrises, MarcaFinNivel);
    while (ColaGrises!= VACIO)
    {
      Sacar (ColaGrises, NodoActual);

      if (NodoActual=MarcaFinNivel) & (ColaGrises!=VACIO)
        Insertar (ColaGrises, MarcaFinNivel);
      else
      {
        for i=0 to 8 do
        {
          EnviarNodo (Hijo (i, NodoActual));

          if (Hijo (i, NodoActual)=GRIS)
            Insertar (ColaGrises, Hijo(i, NodoActual));
        }
      }
    }
  }
}
```

Algoritmo 6.2. Transmisión de la estructura jerárquica.

- *EnviarNodo* transmite toda la información asociada a un nodo del árbol. Esta información constará del *tipo* de nodo, el *número* de planos y la *referencia* a esos planos en la lista auxiliar,
- la función *Insertar* introduce un nuevo elemento en la cola de nodos pendientes,
- *Sacar* obtiene el primer elemento de esa cola de pendientes,
- la función *Hijo (i, NodoActual)* obtiene el hijo que ocupa la posición *i* del nodo *NodoActual*.

```
FUNCTION RecibirNodo (NodoRaiz)
{
  if (NodoRaiz=GRIS)
  {
    CrearNodoInterno (NodoRaiz);
    Insertar (ColaGrises, NodoRaiz);
    Insertar (ColaGrises, MarcaFinNivel);
    while (ColaGrises!= VACIO)
    {
      Sacar (ColaGrises, NodoActual);
      NodoPadre=NodoActual;
      if (NodoActual=MarcaFinNivel) & (ColaGrises!=VACIO)
        Insertar (ColaGrises, MarcaFinNivel);
      else
      {
        for i=0 to 8 do
        {
          RecibirNodo (Hijo (i, NodoActual));
          if (Hijo (i, NodoActual)=GRIS)
          {
            NodoHijo=CrearNodoInterno (Hijo (i,NodoActual));
            Insertar (ColaGrises, Hijo(i, NodoActual));
          }
          else
            NodoHijo=CrearNodoTerminal (Hijo(i,NodoActual));
          EnlazarHijo (NodoPadre, i, NodoHijo);
        }
      }
    }
  }
  else CrearNodoTerminal (NodoRaiz);
}
```

Algoritmo 6.3. Recepción de la estructura jerárquica.

Para la recepción del árbol se utiliza el mismo algoritmo, cambiando *EnviarNodo* por *RecibirNodo*, al que se añaden las instrucciones necesarias para crear los nodos del árbol recibidos y enlazarlos para construir la estructura jerárquica (algoritmo 6.3):

- *CrearNodoTerminal* crea un nuevo nodo terminal con la información del nodo recibido,
- *CrearNodoInterno* crea un nuevo nodo gris con la información recibida,
- *EnlazarHijo (np, i, nh)* enlaza el nodo *nh* como hijo número *i* del nodo *np*.

Este algoritmo deberá estar sincronizado con el de transmisión, de forma que la información que espera recibir sea la que el emisor le envía.

6.3.3. Transmisión y recepción progresiva del modelo completo

El algoritmo completo de transmisión progresiva de un modelo SP-Octree comenzará enviando la información de la caja englobante del modelo, así como el número de niveles de la estructura.

Una vez enviado el nodo raíz del árbol, enviamos la información geométrica de los planos que aparecen en él. Como aún no se ha enviado ningún plano, todos los que aparezcan en ese nodo deben ser enviados. Por ello, la primera llamada a la función ***EnviarGeometría*** tiene como límite del índice en el nivel anterior el valor *-1*.

Si el nodo raíz no es terminal, en cuyo caso ya habría concluido la transmisión), insertamos en la lista de nodos pendientes el nodo raíz y una marca de final de nivel.

A continuación pasamos a transmitir, nivel a nivel, la información de los hijos de los nodos grises pendientes del nivel anterior, y la información geométrica de los planos que aparecen en cada uno de ellos (algoritmo 6.4).

Mientras estemos enviando un nivel, almacenamos el índice del último plano nuevo enviado. Al cambiar de nivel, ese valor será el límite a tener en cuenta para no enviar planos repetidos al transmitir el siguiente nivel.

```
FUNCTION EnviarSPOctree()
{
  Enviar (CajaEnglobante);
  Enviar (NivelesArbol);

  EnviarNodo (NodoRaiz);
  EnviarGeometria(NodoRaiz, -1, IndiceMaximo);

  if (NodoRaiz=GRIS)
  {
    Insertar (ColaGrises, NodoRaiz);
    Insertar (ColaGrises, MarcaFinNivel);
    LimiteNivelAnterior=IndiceMaximo;

    while (ColaGrises!= VACIO)
    {
      Sacar (ColaGrises, NodoActual);

      if (NodoActual=MarcaFinNivel) & (ColaGrises!=VACIO)
      {
        Insertar (ColaGrises, MarcaFinNivel);
        LimiteNivelAnterior=IndiceMaximo;
      }
      else
      {
        for i=0 to 8 do
        {
          EnviarNodo (Hijo (i, NodoActual));
          EnviarGeometria( Hijo (i, NodoActual),
                           LimiteNivelAnterior,
                           IndiceMaximo);

          if (Hijo (i, NodoActual)=GRIS)
            Insertar (ColaGrises, Hijo(i, NodoActual));
        }
      }
    } //while
  }
}
```

Algoritmo 6.4. Transmisión progresiva de un *SP-Octree*.

La recepción del modelo sigue los mismos pasos que el envío, con la particularidad de que el receptor deberá primero inicializar las estructuras necesarias para almacenar el modelo (algoritmo 6.5).

```
FUNCTION RecibirSPOctree()
{
  Inicializar (SP-Octree);
  Inicializar (ListaAuxiliarPlanos);
  Recibir (CajaEnglobante);
  Recibir (NivelesArbol);
  RecibirNodo (NodoRaiz);
  RecibirGeometria(NodoRaiz, -1, IndiceMaximo);
  if (NodoRaiz=GRIS)
  {
    SP-Octree=CrearNodoInterno (NodoRaiz);
    Insertar (ColaGrises, NodoRaiz);
    Insertar (ColaGrises, MarcaFinNivel);
    LimiteNivelAnterior=IndiceMaximo;
    while (ColaGrises!= VACIO)
    {
      Sacar (ColaGrises, NodoActual);
      NodoPadre=NodoActual;
      if (NodoActual=MarcaFinNivel) & (ColaGrises!=VACIO)
      {
        Insertar (ColaGrises, MarcaFinNivel);
        LimiteNivelAnterior=IndiceMaximo;
      }
      else
      {
        for i=0 to 8 do
        {
          RecibirNodo (Hijo (i, NodoActual));
          RecibirGeometria( Hijo (i, NodoActual),
                            LimiteNivelAnterior, IndiceMaximo);
          if (Hijo (i, NodoActual)=GRIS)
          {
            NodoHijo=CrearNodoInterno(Hijo(i,NodoActual));
            Insertar (ColaGrises, Hijo(i, NodoActual));
          }
          else
            NodoHijo=CrearNodoTerminal(Hijo(i,NodoActual));

          EnlazarHijo (NodoPadre, i, NodoHijo);
        }
      }
    } //while
  }
  else SP-Octree=CrearNodoTerminal (NodoRaiz);
}
```

Algoritmo 6.5. Recepción progresiva de un *SP-Octree*.

A medida que recibe la información, el cliente receptor deberá ir construyendo tanto el árbol octal como la lista auxiliar con la información geométrica de los planos a los que se hará referencia desde los nodos del árbol.

Cada vez que sacamos un nodo gris de la lista de nodos pendientes, ese nodo ya estará creado en el árbol al haberse procesado en el nivel anterior. Por tanto, lo que debemos hacer será procesar sus hijos, ir creando los nodos asociados a los mismos, e ir enlazándolos con el nodo antes creado.

6.4. Resultados

Una de las principales ventajas del esquema propuesto es que permite la representación de sólidos definidos no sólo por mallas de triángulos, sino que podemos representar objetos cuya frontera esté formada por cualquier tipo de malla poliédrica, sin necesitar la triangulación de la misma como ocurre con los esquemas de multirresolución y transmisión progresiva actuales.

En las siguientes figuras y tablas mostramos los resultados obtenidos sobre cuatro modelos de sólidos con distinta complejidad (figura 6.5): una pieza mecánica sencilla definida por una malla de 43 polígonos, un modelo de una vaca con 562 triángulos, la cabeza de la vaca anterior con mayor resolución (1076 triángulos) y una escultura de conejo (muy utilizada en esquemas de multirresolución) a baja resolución con 1466 triángulos.

Se han tomado datos sobre el tamaño de la información que se transmite en cada nivel, así como del número de planos de la frontera que se envía por niveles.

Además, en cada etapa de la transmisión se ha obtenido el volumen del modelo transmitido, lo que nos permite obtener una estimación del error cometido en cada nivel comparándolo con el volumen del modelo completo. Este error viene dado por la aparición de nodos grises de último nivel, por lo que el volumen representado por esos nodos será la cota máxima del error cometido en el modelo enviado en cada etapa de la transmisión.

En las tablas 6.3, 6.4 y 6.5 mostramos los resultados para la transmisión del modelo de la pieza mecánica de la figura 6.5.

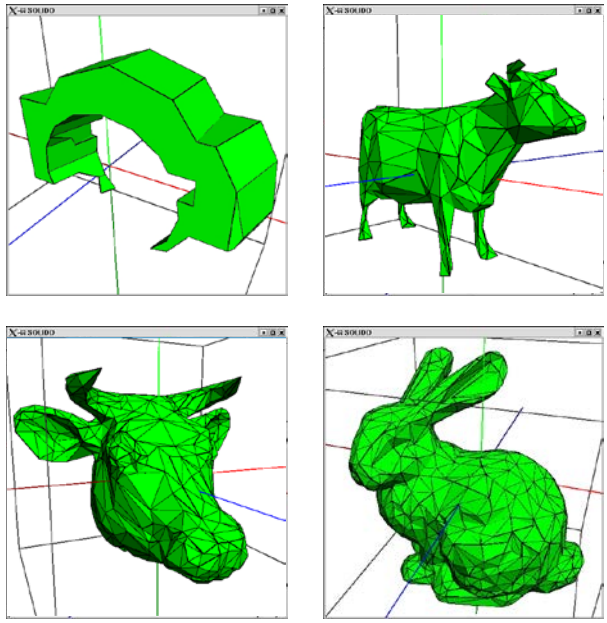


Figura 6.5. Modelos utilizados para las pruebas de transmisión.

En la primera columna de la tabla 6.3 vemos el nivel transmitido, en las tres siguientes mostramos el número total de planos enviados hasta cada nivel, el número de planos transmitidos y el porcentaje de planos enviados en cada nivel. En las tres últimas tenemos el total de información (en bytes) enviada desde el comienzo hasta cada nivel, y el tamaño y el porcentaje de información enviado en cada uno de los niveles.

	Planos transmitidos			Tamaño (Bytes)		
Nivel	**Global**	**Por nivel**	**% por nivel**	**Global**	**Por nivel**	**% por nivel**
0	13	13	30,23%	766	766	15,41%
1	15	2	4,65%	903	137	2,76%
2	19	4	9,30%	1172	269	5,41%
3	35	16	37,21%	2429	1257	25,29%
4	41	6	13,95%	3406	977	19,66%
5	42	1	2,33%	4545	1139	22,92%
6	43	1	2,33%	4970	425	8,55%

Tabla 6.3. *Pieza mecánica*: datos sobre la transmisión por niveles.

En la tabla 6.4 vemos el total de nodos del árbol transmitido en cada nivel, así como el número de nodos grises en el último nivel, y su porcentaje respecto al total de nodos del árbol. Estos nodos son los que hacen que en cada nivel el modelo no sea exacto ya que se trata de nodos que deberán dividirse en el siguiente nivel.

	Nodos del árbol transmitido por niveles		
Nivel	**Total**	**Grises último nivel**	**% Grises último nivel**
0	1	1	100,00%
1	9	2	22,22%
2	25	8	32,00%
3	89	11	12,36%
4	177	16	9,04%
5	305	7	2,30%
6	361	0	0,00%

Tabla 6.4. *Pieza mecánica*: número de nodos por niveles.

En la tabla 6.5 podemos ver el volumen representado en los nodos grises de último nivel, que será la cota máxima del error cometido en el modelo transmitido en cada nivel. Las tres últimas columnas muestran el volumen total del modelo en cada nivel, así como el error cometido en cada uno de ellos.

	Ultimo nivel		Volumen		
Nivel	**Lado nodos último nivel (mm)**	**Volumen grises último nivel (mm^3)**	**Volumen Modelo (mm^3)**	**Error (mm^3)**	**% Error**
0	100,000	119475,123	119475,123	72052,456	151,94%
1	50,000	98613,631	112543,256	65120,590	137,32%
2	25,000	92271,490	106201,115	58778,449	123,95%
3	12,500	19706,137	54657,032	7234,365	15,26%
4	6,250	2876,151	48821,874	1399,208	2,95%
5	3,125	169,334	47472,631	49,965	0,11%
6	1,563	0,000	47422,666	0,000	0%

Tabla 6.5. *Pieza mecánica*: volumen del modelo y error por nivel.

Como vemos, en la cuarta fase de la transmisión, el número de nodos con error (grises de último nivel) es aproximadamente de un 12% del total de nodos transmitidos, lo que se refleja en una aproximación bastante buena del modelo final desde el nivel 3 del modelo.

La gráfica de la figura 6.6 muestra la evolución del error cometido en el volumen del sólido representado en cada nivel, así como el tamaño de la información enviada en cada nivel y el porcentaje de información total que se tiene transmitida en cada nivel.

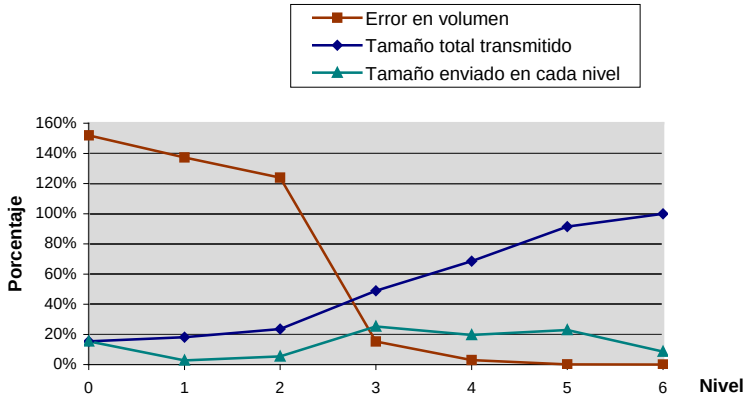


Figura 6.6. *Pieza mecánica*: error y tamaño parcial/total enviado por niveles.

Como podemos observar en la tabla 6.5 y en la figura 6.6, el error cometido en el nivel 3 es de un 15% del volumen total del modelo con tan solo el 50% de información enviada, y en el siguiente nivel, el error es sólo de un 3% con el 70% de la información total transmitida.

La imagen 6.7 muestra la visualización de cada uno de los modelos representados por los árboles transmitidos en cada uno de los 7 niveles del modelo SP-Octree de la pieza mecánica.

En color rojo vemos los planos almacenados en nodos grises de último nivel en cada fase de la transmisión, mientras que en color verde observamos los planos pertenecientes a nodos terminales. En el primer nivel sólo hemos transmitido un árbol con un nodo gris que contiene 13 de los 43 planos que forman la frontera del sólido.

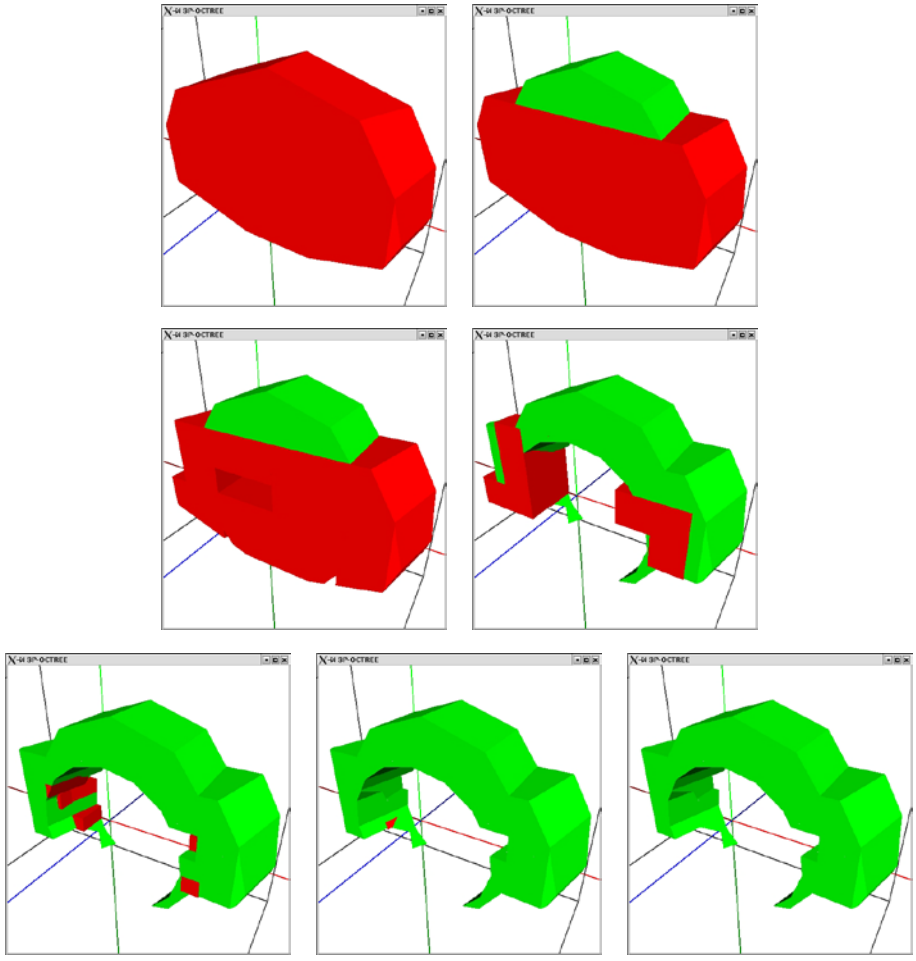


Figura 6.7. ***Pieza mecánica:*** **modelo recibido en cada nivel.**

En las tablas 6.6, 6.7 y 6.8 podemos ver los resultados obtenidos para la transmisión del modelo de la vaca cuya malla está formada por 572 triángulos. En este caso, el SP-Octree obtenido tiene una profundidad de 19 niveles debido a la compleja configuración (concavidades y convexidades) que presentan las caras pequeñas de la zona superior de la cabeza, donde el modelo presenta mayor número de divisiones (figura 6.8).

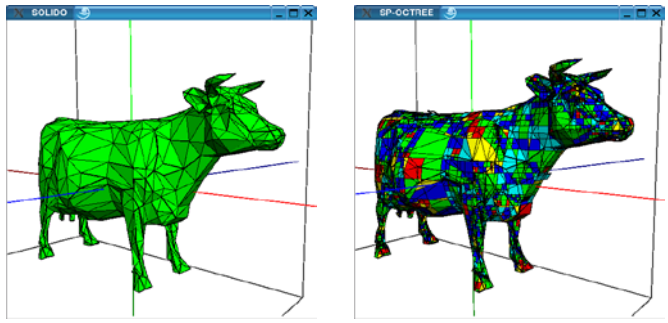


Figura 6.8. ***Vaca:*** **malla poligonal y modelo SP-Octree de 19 niveles.**

En la tabla 6.6 vemos los planos y el total de información transmitida en cada nivel, en la tabla 6.7 aparece la información sobre los nodos grises de último nivel y en la tabla 6.8 la información sobre el volumen del sólido representado en cada árbol completado en cada nivel de la transmisión y el error respecto al volumen del sólido original.

	Planos transmitidos			Tamaño (Bytes)		
Nivel	**Global**	**Por nivel**	**% por nivel**	**Global**	**Por nivel**	**% por nivel**
0	12	12	2,14%	740	740	0,07%
1	31	19	3,38%	1821	1081	0,10%
2	68	37	6,58%	3989	2168	0,21%
3	161	93	16,55%	9783	5794	0,55%
4	385	224	39,86%	27876	18093	1,72%
5	538	153	27,22%	53904	26028	2,48%
6	562	24	4,27%	78556	24652	2,35%
7	562	0	0,00%	96293	17737	1,69%
8	562	0	0,00%	109111	12818	1,22%
9	562	0	0,00%	119609	10498	1,00%
10	562	0	0,00%	132408	12799	1,22%
11	562	0	0,00%	153866	21458	2,04%
12	562	0	0,00%	191715	37849	3,60%
13	562	0	0,00%	264770	73055	6,96%
14	562	0	0,00%	403182	138412	13,18%
15	562	0	0,00%	646664	243482	23,19%
16	562	0	0,00%	976594	329930	31,42%
17	562	0	0,00%	1049806	73212	6,97%
18	562	0	0,00%	1050102	296	0,03%

Tabla 6.6. ***Vaca:*** **datos sobre la transmisión por niveles.**

	Nodos del árbol transmitido por niveles		
Nivel	**Total**	**Grises último nivel**	**% Grises último nivel**
0	1	1	100,00%
1	9	4	44,44%
2	41	18	43,90%
3	185	73	39,46%
4	769	194	25,23%
5	2321	290	12,49%
6	4641	255	5,49%
7	6681	196	2,93%
8	8249	171	2,07%
9	9617	217	2,26%
10	11353	374	3,29%
11	14345	661	4,61%
12	19633	1238	6,31%
13	29537	2293	7,76%
14	47881	3852	8,04%
15	78697	4901	6,23%
16	117905	2930	2,49%
17	141345	11	0,01%
18	141433	0	0,00%

Tabla 6.7. *Vaca*: número de nodos por niveles.

Como vemos, todos los planos del modelo se transmiten en los 7 primeros niveles. Sin embargo, la mayor parte de la información del árbol que almacena el modelo se transmite entre los niveles 13 y 16. Esto se debe a que en este modelo, como podemos ver en la tabla 6.7, el número de nodos *grises* que aparece sufre un gran aumento a partir del nivel 11.

Esto se debe a que en la zona superior de la cabeza de la vaca, los polígonos que forman la frontera del sólido son muy pequeños, por lo que en la mayoría de los casos aparecen nodos con más de un vértice que presentan aristas cóncavas y convexas. Se ha tenido que descender hasta el nivel 17 para conseguir separar esos vértices en nodos convexos o cóncavos, aumentando así el tamaño del árbol en esos niveles.

Al igual que ocurría en el ejemplo anterior, a partir del nivel 6 del árbol, el número de nodos con error (grises de último nivel) es menor del 10% del total de nodos transmitidos, lo que se refleja en que el modelo obtenido en ese nivel presenta un error menor del 1% del volumen total del sólido (tabla 6.8).

	Ultimo nivel		Volumen		
Nivel	**Lado nodos último nivel (mm)**	**Volumen grises último nivel (mm^3)**	**Volumen Modelo (mm^3)**	**Error (mm^3)**	**% Error**
0	104,0335	2,34E+05	235.417,8608	1,84E+05	3,56E+02
1	52,0168	1,53E+05	154.713,0361	1,03E+05	2,00E+02
2	26,0084	1,08E+05	108.035,1775	5,64E+04	1,09E+02
3	13,0042	7,22E+04	84.470,2181	3,28E+04	6,36E+01
4	6,5021	3,01E+04	64.367,4121	1,27E+04	2,47E+01
5	3,2510	6,92E+03	54.456,5878	2,82E+03	5,46E+00
6	1,6255	8,60E+02	51.997,2485	3,62E+02	7,01E-01
7	0,8128	8,78E+01	51.673,4724	3,80E+01	7,36E-02
8	0,4064	9,97E+00	51.639,2884	3,84E+00	7,43E-03
9	0,2032	1,68E+00	51.636,0800	6,31E-01	1,22E-03
10	0,1016	3,65E-01	51.635,5780	1,29E-01	2,49E-04
11	0,0508	8,20E-02	51.635,4789	2,95E-02	5,71E-05
12	0,0254	1,95E-02	51.635,4566	7,20E-03	1,39E-05
13	0,0127	4,50E-03	51.635,4511	1,70E-03	3,30E-06
14	0,0063	9,33E-04	51.635,4498	3,68E-04	7,14E-07
15	0,0032	1,40E-04	51.635,4495	6,51E-05	1,26E-07
16	0,0016	8,12E-06	51.635,4495	6,68E-06	1,29E-08
17	0,0008	1,20E-07	51.635,4494	1,31E-06	2,54E-09
18	0,0004	0,00E+00	51.635,4494	0,00E+00	0,00E+00

Tabla 6.8. *Vaca*: volumen del modelo y error por nivel.

La gráfica de la imagen 6.9 muestra, para el modelo de la vaca, la evolución del error cometido en el volumen del sólido representado en cada nivel, así como el tamaño de la información enviada en cada nivel y el porcentaje de información total que se tiene transmitida en cada nivel.

En la imagen 6.10 vemos la evolución del error (en escala logarítmica para que pueda apreciarse mejor) en el volumen del modelo transmitido en función del porcentaje de información enviada.

En estas gráficas podemos observar cómo, aunque la mayor parte del modelo se transmite en los últimos niveles, el error cometido desciende rápidamente en al transmitir los primeros 6 niveles de la estructura, de forma que con sólo un

20% de la información del modelo transmitida, el receptor recibe una buena aproximación del sólido representado. A partir de ese nivel, la mejora que se obtiene en cada nueva fase de la transmisión es muy pequeña (figura 6.10).

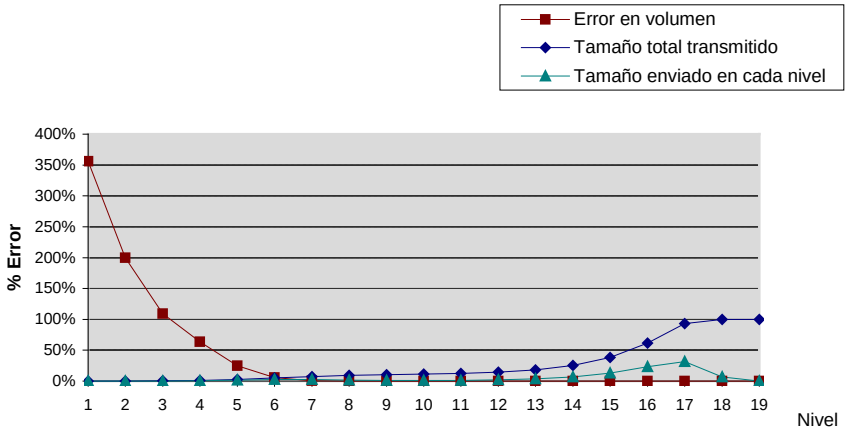


Figura 6.9. *Vaca*: error y tamaño parcial/total enviado por niveles.

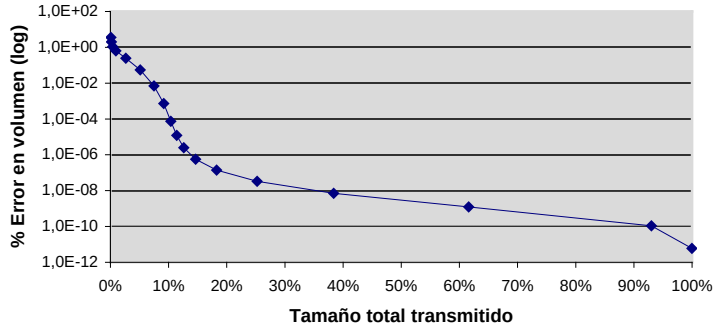


Figura 6.10. *Vaca*: error en volumen por total enviado.

La imagen 6.11 muestra la visualización de cada uno de los modelos representados por los árboles transmitidos en cada uno de los niveles para el modelo de la pieza mecánica. Como vemos, a partir del nivel 7, el número de nodos con error (en rojo) es muy pequeño.

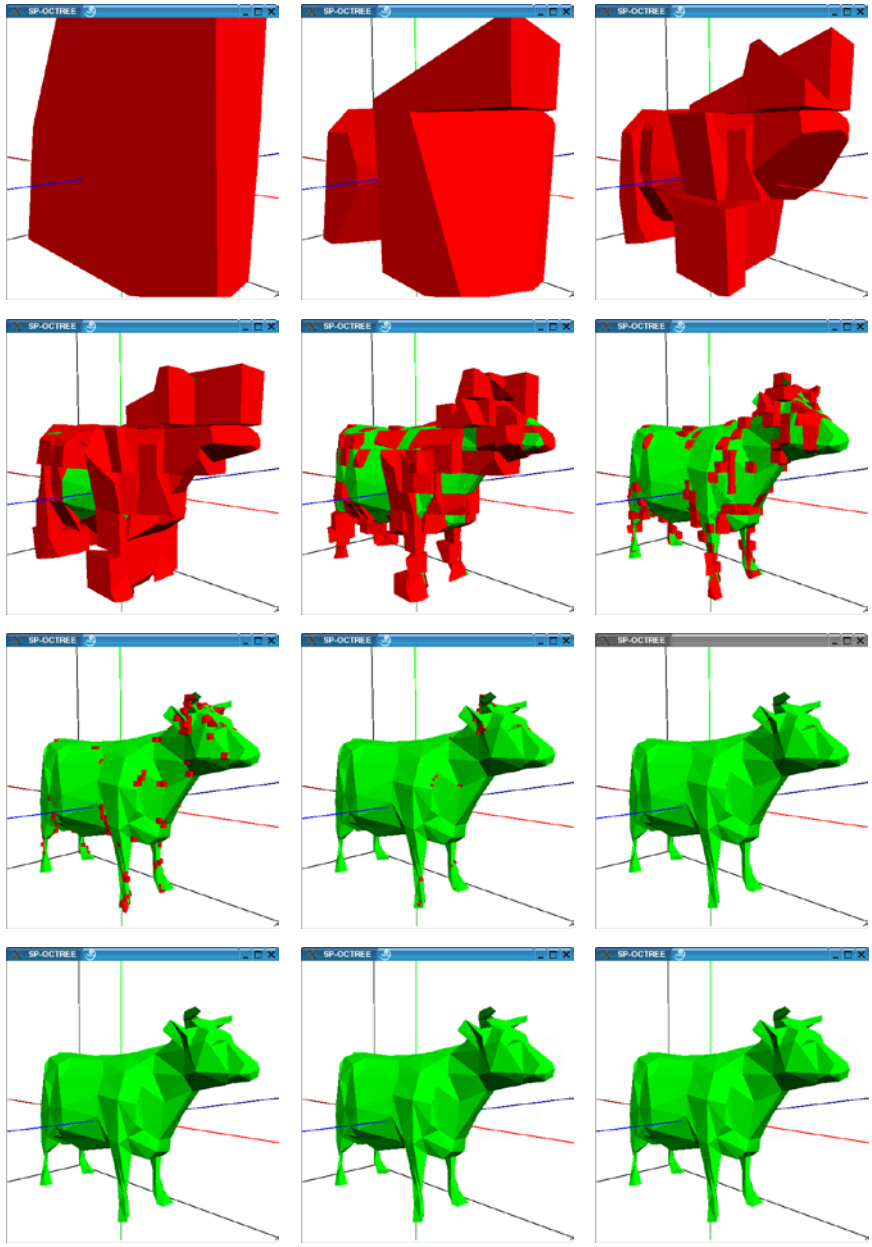


Figura 6.11. ***Vaca:*** **modelo recibido en cada nivel.**

La figura 6.12 muestra los modelos SP-Octrees para los dos últimos sólidos utilizados en las pruebas de transmisión progresiva. Para el modelo de la cabeza de vaca (1076 caras) tenemos un árbol de 17 niveles, mientras que para el conejo (1466 caras), el árbol tiene sólo 15 niveles.

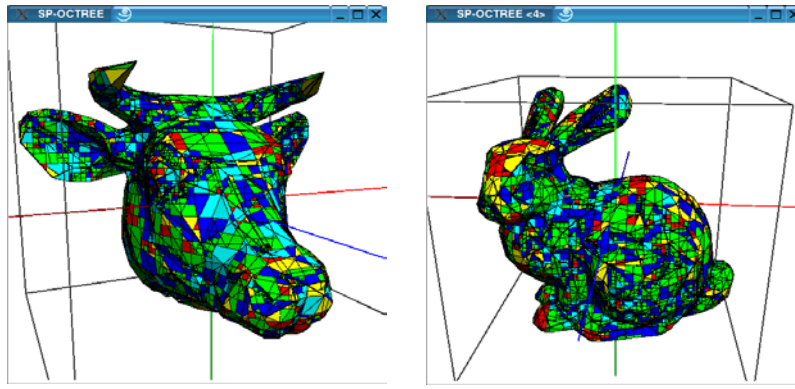


Figura 6.12. Modelos SP-Octree para *cabeza de vaca* (17 niveles) y *conejo* (15 niveles).

En las tablas 6.9 y 6.10 tenemos los datos de la transmisión para el modelo de la cabeza. Las tablas 6.11 y 6.12 muestran esos mismos resultados para la transmisión del modelo del conejo.

En las figuras 6.13 y 6.14 podemos ver gráficamente la relación entre el error cometido en la representación del sólido en cada nivel con la cantidad de información enviada en ese nivel para los dos modelos.

La figura 6.15 muestra para los dos modelos la diferencia de volumen entre el sólido representado por la parte del árbol recibida en función del porcentaje de información transmitida del modelo.

En ambos casos se vuelve a reproducir la misma situación que en los dos ejemplos anteriores: sólo con transmitir un 20% de la información del SP-Octree, el modelo recibido presenta un error menor del 20% respecto al volumen total del sólido original.

	Planos transmitidos			Tamaño (Bytes)		
Nivel	**Global**	**Por nivel**	**% por nivel**	**Global**	**Por nivel**	**% por nivel**
0	42	42	3,90%	2414	2414	1,06%
1	71	29	2,70%	4056	1642	0,72%
2	162	91	8,46%	9408	5352	2,34%
3	324	162	15,06%	19936	10528	4,61%
4	635	311	28,90%	45846	25910	11,35%
5	1032	397	36,90%	105255	59409	26,02%
6	1075	43	4,00%	154832	49577	21,71%
7	1076	1	0,09%	189811	34979	15,32%
8	1076	0	0,00%	208044	18233	7,99%
9	1076	0	0,00%	218137	10093	4,42%
10	1076	0	0,00%	223127	4990	2,19%
11	1076	0	0,00%	225549	2422	1,06%
12	1076	0	0,00%	227073	1524	0,67%
13	1076	0	0,00%	228017	944	0,41%
14	1076	0	0,00%	228244	227	0,10%
15	1076	0	0,00%	228291	47	0,02%
16	1076	0	0,00%	228335	44	0,02%

Tabla 6.9. ***Cabeza de vaca:*** **datos sobre la transmisión por niveles.**

Nivel	Total Nodos	Nodos Grises último nivel	% Nodos Grises último nivel	Volumen nodos grises de último nivel (mm^3)	% Error en volumen
0	1	1	100,00%	4,52E+05	2,56E+02
1	9	8	88,89%	3,49E+05	1,74E+02
2	73	34	46,58%	2,35E+05	8,45E+01
3	345	116	33,62%	1,18E+05	3,29E+01
4	1273	365	28,67%	5,49E+04	1,45E+01
5	4193	604	14,40%	1,26E+04	2,81E+00
6	9025	521	5,77%	1,42E+03	3,47E-01
7	13193	282	2,14%	9,97E+01	2,60E-02
8	15449	161	1,04%	7,34E+00	2,33E-03
9	16737	80	0,48%	4,46E-01	1,14E-04
10	17377	40	0,23%	2,87E-02	6,37E-06
11	17697	25	0,14%	2,26E-03	5,82E-07
12	17897	14	0,08%	1,76E-04	5,49E-08
13	18009	6	0,03%	9,40E-06	2,41E-09
14	18057	1	0,01%	2,23E-07	1,81E-10
15	18065	1	0,01%	2,80E-08	2,36E-11
16	18073	0	0,00%	0,00E+00	0,00E+00

Tabla 6.10. ***Cabeza de vaca*****: número de nodos y error por niveles.**

	Planos transmitidos			Tamaño (Bytes)		
Nivel	**Global**	**Por nivel**	**% por nivel**	**Global**	**Por nivel**	**% por nivel**
0	66	66	4,50%	3740	3740	1,25%
1	155	89	6,07%	8735	4995	1,67%
2	317	162	11,05%	18268	9533	3,19%
3	534	217	14,80%	32463	14195	4,75%
4	1007	473	32,26%	69099	36636	12,27%
5	1437	430	29,33%	145672	76573	25,64%
6	1466	29	1,98%	209323	63651	21,31%
7	1466	0	0,00%	250832	41509	13,90%
8	1466	0	0,00%	274357	23525	7,88%
9	1466	0	0,00%	286458	12101	4,05%
10	1466	0	0,00%	293571	7113	2,38%
11	1466	0	0,00%	296371	2800	0,94%
12	1466	0	0,00%	297754	1383	0,46%
13	1466	0	0,00%	298523	769	0,26%
14	1466	0	0,00%	298696	173	0,06%

Tabla 6.11. ***Conejo:*** **datos sobre la transmisión por niveles.**

Nivel	**Total Nodos**	**Nodos Grises último nivel**	**% Nodos Grises último nivel**	**Volumen nodos grises de último nivel (mm^3)**	**% Error en volumen**
0	1	1	100,00%	1,84E+06	1,43E+02
1	9	8	88,89%	1,41E+06	8,66E+01
2	73	42	57,53%	1,03E+06	3,63E+01
3	409	157	38,39%	6,37E+05	2,22E+01
4	1665	504	30,27%	2,90E+05	1,02E+01
5	5697	781	13,71%	6,23E+04	2,28E+00
6	11945	605	5,06%	5,93E+03	2,59E-01
7	16785	357	2,13%	3,71E+02	1,74E-02
8	19641	189	0,96%	2,36E+01	1,21E-03
9	21153	106	0,50%	1,47E+00	5,45E-05
10	22001	41	0,19%	1,23E-01	4,52E-06
11	22329	23	0,10%	7,79E-03	2,53E-07
12	22513	11	0,05%	5,77E-04	1,99E-08
13	22601	5	0,02%	1,14E-05	9,91E-11
14	22641	0	0,00%	0,00E+00	0,00E+00

Tabla 6.12. ***Conejo:*** **nodos transmitidos y error por niveles.**

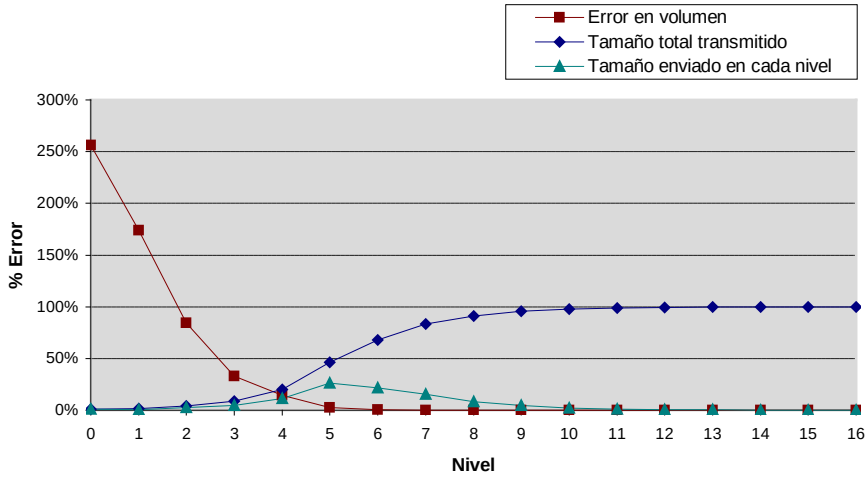


Figura 6.13. *Cabeza de vaca*: error y tamaño transmitido por niveles.

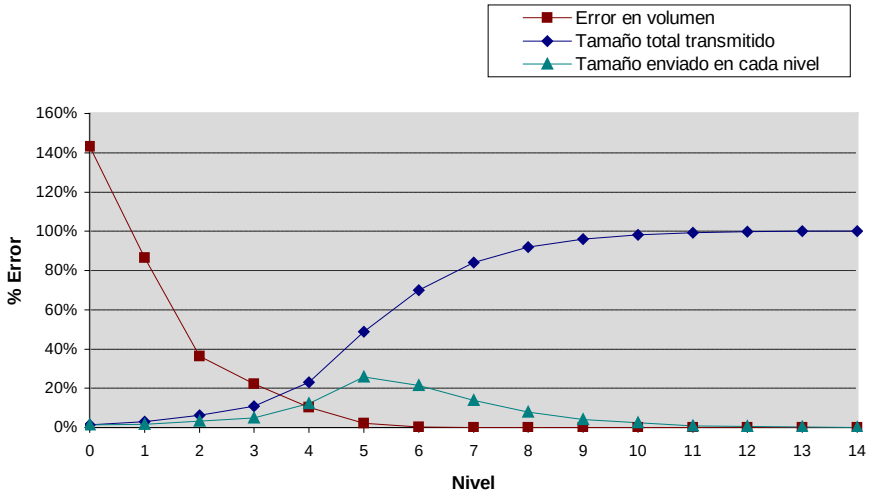


Figura 6.14. *Conejo:* error y tamaño transmitido por niveles.

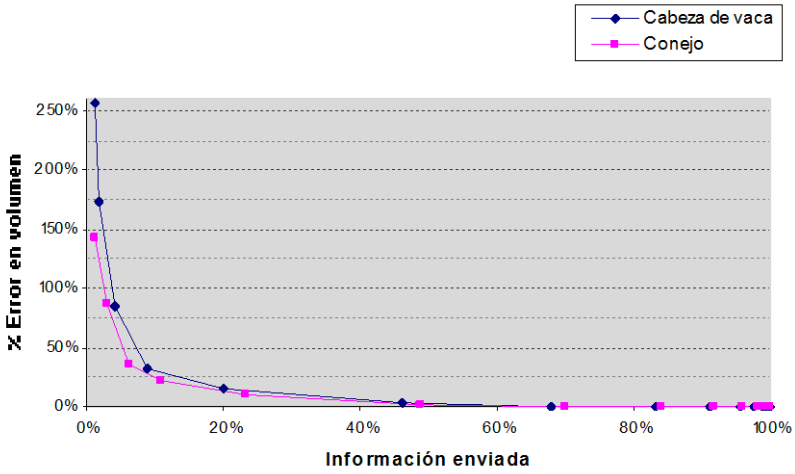


Figura 6.15. Error en volumen por total enviado para modelos de *conejo* y *cabeza de vaca*.

En las imágenes 6.15 y 6.16 vemos los once primeros niveles de la transmisión del modelo de la cabeza de vaca y del conejo respectivamente. Como en los anteriores ejemplos, mostramos en color rojo los planos que aparecen en nodos grises de último nivel en cada fase de la transmisión, y en color verde los referenciados en nodos terminales.

Aunque los árboles de los modelos completos tienen 17 y 15 niveles respectivamente, el porcentaje de nodos grises a partir del nivel 9 en ambos casos hace que la diferencia entre el modelo transmitido en esos niveles sea muy pequeña. Como vemos, en cada nivel de la transmisión obtenemos una representación aproximada, cada vez con mejor resolución, del modelo final.

Podemos observar en todos los ejemplos que el receptor tiene desde el primer momento una aproximación del sólido representado, que irá mejorando conforme avancemos en la transmisión.

Esta aproximación le puede permitir, por ejemplo, definir un nuevo punto de vista o seleccionar áreas específicas para que la transmisión continúe enviando sólo ramas concretas del árbol [Hopp97]. De esta forma podríamos definir mecanismos de transmisión progresiva adaptativa en función de las necesidades del receptor.

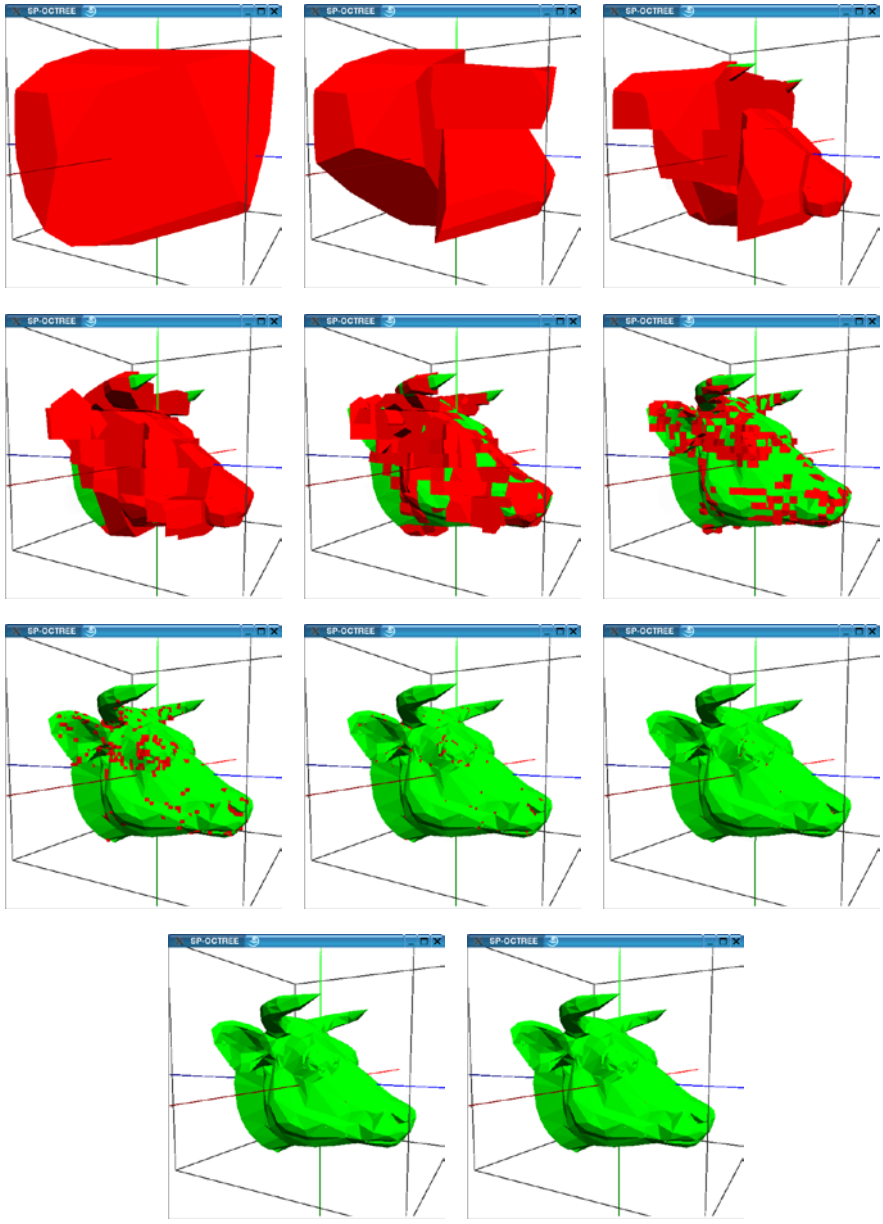


Figura 6.16. ***Cabeza de vaca:*** **modelo recibido en cada nivel.**

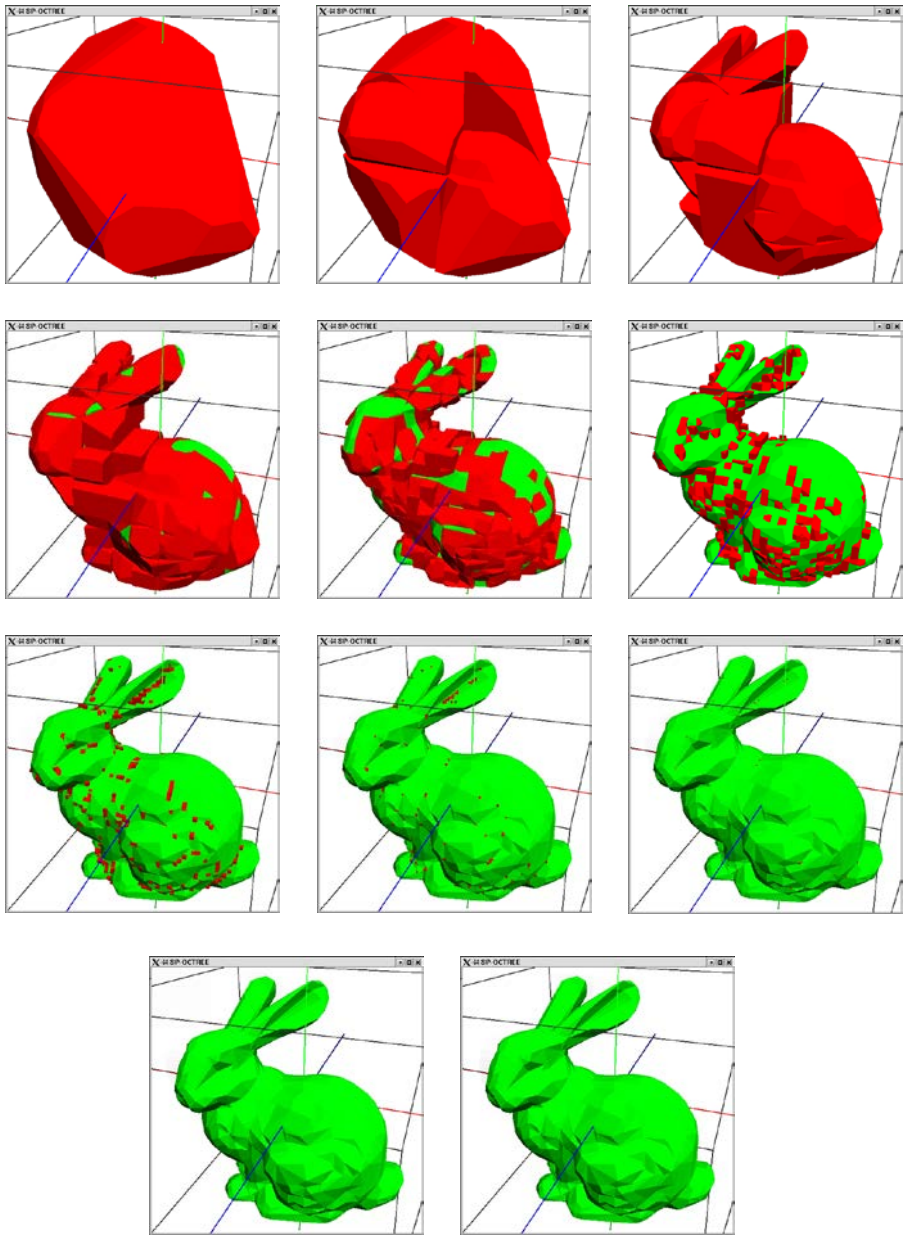


Figura 6.17. ***Modelo de conejo:*** **transmisión progresiva por niveles.**

6.5. Conclusiones

Es este capítulo hemos presentado el uso del esquema propuesto en una de las aplicaciones más necesarias en el modelado de sólidos en entornos distribuidos: la transmisión progresiva del modelo.

La propia estructura jerárquica del esquema y el almacenamiento de la información geométrica sin redundancias hace posible una transmisión del modelo por niveles, que permite al receptor trabajar con una representación aproximada del modelo desde el primer momento. Hemos presentado los algoritmos utilizados para transmitir y recibir de forma progresiva tanto la información geométrica del modelo como la estructura jerárquica que lo soporta.

Debemos notar que en ningún momento se necesita transmitir información geométrica de los polígonos que forman la frontera del sólido, sino sólo las ecuaciones de los planos que soportan esos polígonos. Con el algoritmo propuesto para realizar la transmisión, el cliente que recibe el modelo deberá reconstruir, si lo necesita, la geometría de la frontera del sólido a partir del conjunto de planos y de la estructura jerárquica recibida en cada nivel.

Otra ventaja del esquema es que nos permite representar no solo objetos definidos por mallas de triángulos, sino que podemos aplicar el mecanismo de transmisión propuesto para cualquier tipo de malla poliédrica.

Por último hemos presentado ejemplos concretos de la utilización del mecanismo propuesto, estudiando los resultados obtenidos en cuanto a tamaño de la transmisión en cada nivel, número de planos enviados en cada fase de la transmisión y error cometido en los modelos intermedios.

CAPÍTULO

7

Conclusiones y Trabajos Futuros

7

Finalizamos la memoria con un resumen de las conclusiones obtenidas y de las principales aportaciones desarrolladas en este trabajo. Además, presentamos algunos de los aspectos que han quedado abiertos a partir del trabajo que aquí hemos presentado y que definen las líneas de trabajo futuro que pretendemos continuar a partir de ahora o que, en algunos casos, ya estamos desarrollando.

7.1. Conclusiones y principales aportaciones

En el presente trabajo se ha presentado un nuevo esquema de representación de sólidos, *SP-Octree*, basado en la modificación de los modelos *Octree* clásico y *Octree Extendido*, introduciendo parte de la información de la frontera del objeto representado, tanto en los nodos terminales como en los nodos internos [Cano00] [Cano02].

Se han descrito los distintos tipos de nodos utilizados, la estructura interna utilizada para almacenar el modelo y el proceso de construcción del mismo a partir de objetos representados por su frontera. Además, se ha evaluado el modelo en cuanto al espacio necesario requerido, comparándolo con los *Octrees Extendidos.*

El método propuesto permite una representación exacta de objetos poliédricos. Al poder almacenar la información de los planos que definen la frontera del sólido en una estructura auxiliar al propio árbol que representa el modelo, y al reducir la repetición de planos en nodos vecinos que los comparten insertándolos en los nodos interiores de la estructura, las necesidades de espacio son menores que en los Octrees clásicos y las extensiones propuestas.

Además, seguimos manteniendo las propiedades de ordenación de estos últimos, y debido a la propia orientación de los planos insertados en cada nodo, es fácil hacer la interrogación y visualización del modelo. Se han diseñado los algoritmos básicos para realizar algunas de las operaciones más interesantes y necesarias en modelado de sólidos utilizando el esquema propuesto.

En concreto, se ha implementado la clasificación de puntos respecto al modelo, la visualización del mismo, tanto de forma completa como de forma adaptativa y se han descrito los algoritmos necesarios para convertir los modelos entre nuestro esquema y algunos de los más utilizados en modelado de sólidos.

El número de polígonos necesarios para visualizar el modelo con el esquema propuesto es menor que el necesario en los *Octrees Extendidos*, debido a que en estos últimos los planos se encuentran divididos en una serie de nodos vecinos, mientras que en el esquema que proponemos, si forman parte de la envolvente convexa de la parte del sólido perteneciente a cada nodo, no deberá dividirse entre sus nodos hijos.

Esto se debe a que el número de nodos que aparecen en el esquema presentado depende sólo de las aristas cóncavas que aparezcan en el sólido, mientras que en los *Octrees Extendidos* dependen del número de vértices y aristas del mismo.

Otras de las operaciones más importantes en todo esquema de modelado de sólidos son las operaciones booleanas. Se han presentado también los algoritmos necesarios para obtener el *complementario* de un modelo SP-Octree y la *intersección* entre modelos, así como la implementación de la *unión* y *diferencia* en base a estas operaciones.

Por ultimo, se ha presentado la utilización del esquema diseñado para la transmisión progresiva del modelo [Cano03]. La propia estructura jerárquica utilizada permite enviar el modelo nivel a nivel de forma que el receptor puede visualizar el modelo y trabajar sobre él desde el comienzo de la transmisión al recibir, en cada fase de la transmisión, una representación válida aproximada del objeto modelado.

Se han presentado los algoritmos necesarios para realizar la recepción/transmisión de modelos SP-Octree y se ha realizado un estudio del proceso en función del volumen de información enviada/recibida en cada fase de la transmisión.

7.2. Trabajos futuros

En base al trabajo desarrollado y presentado en esta memoria, quedan aún abiertas una serie de líneas sobre las que pretendemos continuar nuestro trabajo, o sobre las que intentaremos aplicar el esquema propuesto para comprobar su validez y utilidad.

En algunos casos, ya se ha comenzado a trabajar en ellas y se han obtenido algunos resultados. Enumeramos a continuación las principales tareas futuras planteadas:

- Mejorar el esquema para conseguir optimizar los algoritmos aquí descritos. Muchos de ellos presentan determinados problemas por la no utilización de geometría explícita en los nodos de la estructura o por la utilización de nodos *vértice*, cuya gestión es complicada.

  Estamos estudiando la utilización de árboles BSP en los nodos y la eliminación/optimización del tipo de nodo *vértice* para mejorar los algoritmos en los que actualmente influye negativamente.

- Modificar la información almacenada en los nodos, con idea de evaluar cómo afectaría en espacio y tiempo para los algoritmos la inclusión de geometría explícita en los nodos, en vez de los planos actuales.

  Intentaremos definir una heurística que permita decidir en que casos puede ser interesante almacenar dicha información.

- Estudiar e implementar los cambios necesarios en el esquema para permitir representar sólidos *no-variedad*.

- Estudiar la posibilidad de utilizar el esquema para sólidos cuya frontera no esté formada por caras planas.

- Mejorar la visualización adaptativa en función del punto de vista del observador, zonas de interés, etc. Aplicar mecanismos que permitan eliminar los agujeros producidos en el algoritmo presentado de forma eficiente.

- Estudiar la forma de balancear los árboles SP-Octree para mejorar el proceso de transmisión progresiva y acelerar los cálculos sobre el modelo.

- Edición progresiva que permita trabajar con el modelo a medida que se va realizando la transmisión por niveles del mismo.

- Estudio del esquema propuesto como método de indexación espacial [Vaně91, Fuji85] para acelerar las operaciones sobre modelos B-Rep complejos, aplicándolo a problemas de visibilidad de grandes modelos.

- Integración del esquema en un sistema interactivo de modelado de sólidos que permita evaluar el esquema.

Bibliografía

B

A continuación indicamos las referencias bibliográficas más importantes utilizadas para el desarrollo de esta memoria y de todo el trabajo reflejado en la misma.

[Andu96] Andujar, C.; Ayala, D.; Brunet, P.; Joan-Arinyo, R.; Sole, J.: *"Automatic generation of multiresolution boundary representations"*. Computer Graphics Forum, 15 (3), pp: 87-96 (1996).

[Ayal85] Ayala, D.; Brunet, P.; Joan, R.; Navazo, I.: *"Object representation by means of nonminimal division of quadtrees and octrees"*. ACM Transactions on Graphics, 4(1), pp: 41-59 (1985).

[Ayal88] Ayala, D.: *"Boolean operations between solids and surfaces by octrees: models and algorithms"*. Computer Aided Design, 20(8), pp: 452-465, (1988).

[Ayal91] Ayala, D.; Battle, F.; Brunet, I.; Navazo, I.: *"Boolean Operations between Extended Octrees"*. Documento de trabajo LSI-91-31, Dpto. Lenguajes y Sistemas Informáticos, Universidad Politécnica de Cataluña, Barcelona (1991).

[Ayal97] Ayala, D.; Brunet, P.; Joan, R.; Navazo, I.: *"Multiresolution approximation of polyhedral solids"*. En Roller, D. and Brunet, P. (eds.): *CAD Systems Development: Tools and Methods*, pp: 327-343. Springer-Verlag, (1997).

[Baer79] Baer, A.; Eastman, C.; Henrion, M.: "*Geometric Modeling: a survey"*. Computer Aided Design, 11(5), pp: 253-272 (1979).

[Baum75] Baumgart, B.: "*A polyhedron representation for computer vision"*. National Computer Conference, pp: 589-596, AFIPS Conf. Proc. 1975.

[Bron87] Bronsvoort, W.F.: "*Geometric Modeling"*. Tutorial Eurographics (1987).

[Bron88] Bronsvoort, W.F.; Post, F.H.: "*Geometric Modeling"*. En *Advance in Computer Graphics III*, Ed. Ruiter, (1988).

[Brun85] Brunet, P.; Navazo, I.: "*Geometric modelling using exact octree representation of polyhedral objects"*. Eurographics'85, pp: 159-169 (1985).

[Brun87] Brunet, P.; Ayala, D.: "*Extended octree representation of free form surfaces"*. Computer Aided Geometric Design, 4, pp: 141-154, (1987).

[Brun90a] Brunet, P.; Navazo, I.: "*Solid Representation and Operation Using Extended Octrees"*. ACM Transactions on Graphics, Vol. 9, nº 2, pp: 170-197 (1990).

[Brun90b] Brunet, P.: "*Face Octrees. Envolved Algorithms and Applications"*. Documento de trabajo LSI-90-14, Dpto. Lenguajes y Sistemas Informáticos, Universidad Politécnica de Cataluña, Barcelona (1990).

[Brun92] Brunet, P.; Juan, R.; Navazo. I.: "*Octree representations in solid modelling"*. En Zobrist, G. (ed.): *Progress in Computer Graphics*, pp: 164-215 (1992).

[Cano96] Cano, P; Velasco, F.; Torres, J.C.: "*Modelado 2D mediante jerarquía de Quadtrees en distintos sistemas de coordenadas"*. II Jornadas de Informática, pp: 153-162. Almuñecar, Granada (1996).

[Cano98] Cano, P.; Velasco, F.; León, A.: "*Modelado y visualización de volúmenes"*. *I* Jornadas de Informática Gráfica, pp: 21-30. Granada, (1998).

[Cano00] Cano, P.; Torres, J.C.: "*Octrees Modificados: una extensión a Octrees clásicos"*. II Jornadas Andalusíes de Informática Gráfica, pp: 1-10, Sevilla, (2000).

[Cano02] Cano, P.; Torres, J.C.: *"Representation of Polyhedral Objects using SP-Octrees"*. Journal of WSCG, vol. 10 (1) pp: 95-101 (2002).

[Cano03] Cano, P.; Torres, J.C., Velasco, F.: *"Progressive transmission of Polyhedral Solids using a Hierarchical Representation Scheme"*. Journal of WSCG, vol. 11 (1) pp: 95-101 (2003).

[Chen88] Chen, H.H.; Huang, T.S.: *"A Survey of Construction and Manipulation of Octrees"*. Computer Vision, Graphics and Image Processing, 43, pp: 409-431 (1988).

[Care97] Carey, R.; Bell, G.; Marrin, C.: *"The Virtual Reality Modeling Language"* ISO/IEC DIS14772-1. http://www.web3d.org/Specifications (1997).

[Carl85] Carlbom, I.; Chakravarty,I.; Vanderschel, D.A.: *"A hierarchical data structure for representing the spatial decomposition of 3D objects*". IEEE Computer Graphics & Applications, 5(4), pp: 24-31 (1985).

[Comb96] Comba, J.; Taylor, B.: *"Conversion of Binary Space Partitioning Trees to Boundary Representation"*. Proceedings of Theory and Practice of Geometric Modeling, Tuebingen, Germany (1996).

[Dieh88] Diehl, R.: *"Conversion of Boundary Representations to Bintrees"*. Eurographics'88, pp: 117-127, (1988).

[Dobk88] Dobkin, D.; Guibas, L.; Hershberger, J.; Snoeyink, J.: *"An Efficient Algorithm for Finding the CSG Representation of a Simple Polygon"*. Computer Graphics 22(4), pp. 31-40 (1988).

[Doct81] Doctor, L. J.; Torborg, J. G.: *"Display techniques for octree-encoded objects"*. IEEE Computer Graphics and Applications, 1 (3), pp: 29-38 (1981).

[Durs89] Durst, M.J.; Kunii, T.L.: *"Integrated Polytrees: A Generalized model for integrating spatial decomposition and boundary representation"*. En Strasser, W. (Ed.): *Theory and Practice of Solid Modelling.* Springer Verlag, (1989).

[Elbe88] Elber, G.; Shpitalni, M.: *"Octree creation via C.S.G. definition"*. The Visual Computer, 4, pp. 53-64 (1988).

[Feit98] Feito, F.; Torres, J.C.: *"Inclusion test for general polyhedra"*. Computer & Graphics, vol. 21, nº 1, (1998).

[Fole96] Foley J.D.; van Dam A.; Feiner S.K.; Hughes J.F.: "*Computer Graphics. Principles and Practice, 2º Edition in C"*. Addison-Wesley, (1996).

[Fuch80] Fuchs, H.; Kedem, Z.; Naylor, B.: "*On Visible Surface Generation by a Priori Tree Structures"*. ACM Computer Graphics, 14(3), pp: 124-133, (1980).

[Fuch83] Fuchs, H.; Abram, G.D.; Grant, E.D.: "*Near Real-Time Shaded Display of Rigid Objects"*. ACM Computer Graphics, 17(3), pp: 65-72, (1983).

[Fuji84] Fujimura, K.; Yamaguchi, K.; Kunii, T.: "*Octree-related data structures and algorithms"*. IEEE Computer Graphics and Applications, 4, pp: 53-59, (1984).

[Fuji85] Fujimura, K.; Kunii, T.: "*A Hierarchical Space Indexing Method"*. En Kunii, T. (ed.): *Computer Graphics: Visual technology and Art*, Proceedings of Computer Graphics Tokyo '85 Conference, pp. 21-34. Ed. Springer, (1985).

[Garg82a] Gargantini, I.: "*Linear octrees for fast processing of three-dimensional objects"*. Computer Graphics and Image Processing, 20, pp: 356-374 (1982).

[Garg82b] Gargantini, I.: "*An effective way to represent quadtrees"*. Communications of ACM, 25(12), pp: 905-910 (1982).

[Garg86] Gargantini, I.; Walsh, T.R.; Wu, O.L.: "*Viewing transformations of voxel-based objects via linear octrees"*. IEEE Computer Graphics and Applications, 6(10), pp: 12-21 (1986).

[Garg89] Gargantini, I.; Schrack, G.; Atkinson, H.H.: "*Adaptive display of linear octrees"*. Computer & Graphics, 13(3), pp: 337-343 (1989).

[Garl99] Garland, M.: "*Multiresolution Modelling: Survey & Future Opportunities"*. EUROGRAPHICS'99 State of the Art Report (1999).

[Gurs91] Gursoz, E.; Choi, Y.; Prinz, F.: "*Boolean set operations on non-manifold boundary representation objects"*. Computer Aided Design, vol. 23, nº 1, (1991),

[Heck94] Heckbert, P.S.; Garland, M.: "*Multiresolution Modeling for Fast Rendering"*. Graphics Interface'94, pp: 43-50 (1994).

[Hoff89] Hoffmann, C.M.: *"Geometric and Solid Modeling: An Introduction"*. Morgan Kaufmann, 1989.

[Hoff89b] Hoffmann, C.M.: *"The problems of accuracy and robustness in geometric computation"*. IEEE Computer Graphics and Applications, 22(3), pp: 31-41 (1989).

[Hopp93] Hoppe H.; DeRose, T.; Duchamp, T.; McDonald, J.; Stuetzle, W.: *"Mesh Optimization"*. Proceedings SIGGRAPH 93, pp: 19–26 (1993).

[Hopp96] Hoppe H.: *"Progressive meshes"*. Proceedings SIGGRAPH 96, pp: 99–108. ACM SIGGRAPH (1996).

[Hopp97] Hoppe H.: *"View-Dependent Refinement of Progressive Meshes"*. Proceedings SIGGRAPH 97, ACM SIGGRAPH (1997).

[Horn89] Horn, W.; Taylor, D.: *"A theorem to determine the spatial containment of a point in a planar polyhedron"*. Computer Vision Graphics and Image Processing, 45 (1989).

[Jack80] Jackins, CH.; Tanimoto, S.L.: *"Octrees and their use in representing three-dimensional objects"*. Computer Graphics and Image Processing, 14, pp: 249-270 (1980).

[Joan89] Joan-Arinyo, R.: *"On Boundary to CSG and Extended Octrees to CSG Conversions"*. En W. Strasser (ed.): *Theory and Practice of Geometric Modeling*, pp: 349-367. Springer-Verlag, (1989).

[Joan96] Joan, R.; Vinacua, A.; Brunet, P.: *"Classification of a Point with Respect to a Polyhedron Vertex"*. International Journal of Computational Geometry and Applications, 6(2), pp: 157-167, (1996).

[Kala82] Kalay, Y.E.: *"Determining the spatial containment of a point in general polyhedra"*. Computer Graphics Image Processing, 19, pp: 303-334 (1982).

[Kauf94] Kaufman, A.: *"Trends in volume visualization and volume graphics"*. Scientific Visualization, pp: 3-19. Ed. Academic Press Ltd., (1994).

[Kobb00a] Kobbelt, L.P.; Bischoff, S.; Botsch, M.; Kähler, k.; Rössl, C.; Schneider, R.; Vorsatz, J.: *"Geometric Modeling Based on Polygonal Meshes"*. Eurographics Tutorial (2000).

[Kobb00b] Kobbelt, L.P.; Bareuther, T.; Seidel, H.P.: "Multiresolution Shape Deformations for Meshes with Dynamic Vertex Connectivity". Computer Graphics Forum, 19 (3), (2000).

[Kuni85] Kunii, T.L.; Satoh, T.; Yamaguchi, K.; *"Generation of Topological Boundary Representations from Octree Encoding"*, IEEE Computer Graphics and Applications, 5, pp: 27-31, (1985).

[Lane84] Lane, J.; Magedson, D.; Rarick, M.: *"An efficient point in polyhedron algorithm"*. Computer Vision Graphics and Image Processing, 26 (1984).

[Lore87] Lorensen, W.E., Cline, H.E.: "*Marching Cubes: A high resolution 3D surface construction algorithm"*. Computer & Graphics, 21 (4), pp: 163-169 (1987).

[Mänt86] Mäntylä, M.: "*Boolean operations of 2-manifolds trough vertex neighbourhood classification". ACM Transactions on Graphics, 5(1), pp: 1-29*, (1986).

[Mänt88] Mäntylä, M.: "*An introduction to solid modeling"*. Computer Science Press, (1988).

[Mänt89] Mäntylä M.: "*Advanced Topics in Solid Modeling"*. Eurographics'89 Tutorial, (1989).

[Meag80] Meagher, D.: "*Octree encoding: a new technique for the representation, manipulation and display of arbitrary three dimensional objects by computer"*. Technical Report IPL-TR-80-111, Polytechnic Inst., Revisselaer (1980).

[Meag82] Meagher, D.: "*Geometric modelling using octree encoding"*. Computer Graphics and Image Processing, 19(2), pp: 129-147 (1982).

[Mort85] Mortenson, M.E.: "*Geometric Modelling"*. John Wiley & Sons, (1985).

[Nava86] Navazo, I.; Ayala, D.; Brunet, P.: "*A geometric modeller based on the exact octree representation of polyhedra"*. Computer Graphics Forum, 5(2), pp: 91-104 (1986).

[Nava87] Navazo, I.; Fontdecaba, J.; Brunet, P.: "*Extended octrees, between CSG and boundary representations"*. Eurographics'87, pp: 239-247 (1987).

[Nava89] Navazo, I.: *"Extended octree representation of general solids with plane faces: Model structure and algorithms"*. Computer and Graphics, 13, pp: 5-16 (1989).

[Nayl90a] Naylor, B.: *"Binary Space Partitioning Trees as an Alternative Representation of Polytopes"*. Computer Aided Design, 22(4), pp: 250-252, (1990).

[Nayl90b] Naylor, N.; Amanatides, J.; Thibault, W.: *"Merging BSP Trees Yields Polyhedral Set Operations"*. ACM Computer Graphics, 24(4), pp: 115-124, (1990).

[Neid97] Neider, J.; Davis, T.; Woo, M.: *"OpenGL Programming Guide. 2nd edition"*. Addison Wesley, Massachusetts, (1997).

[Oliv84] Oliver, M.A.: *"Two Display Algorithms for Octrees"*. Proceedings of Eurographics'84, pp: 251-265 (1984).

[Paja00] Pajarola, R.; Rossignac, J.: *"Compressed Progressive Meshes"*. IEEE Transactions on Visualization and Computer Graphics, 6(1), pp: 79-93, (2000).

[Pate90] Paterson, M.S.; Yao, F.F.: *"Optimal Binary Space Partitions for Orthogonal Objects"*. Proceedings of 1st Symposium on Discrete Algorithms, pp: 100-106 (1990).

[Pla93] Pla, N.: *"Boolean operations and spatial complexity of face octrees"*. Eurographics'93, Computer Graphics Forum, 12(3), pp:153-164, (1993).

[Pupp97] Puppo, E.: *"Simplification, LOD and Multiresolution Principles and Applications"*. Eurographics'97 Tutorial (1997).

[Putn86] Putnam, L.K.; Subrahmanyam, P.A.: *"Boolean Operations on n-Dimensional Objects"*. IEEE Computer Graphics and Applications, 6, pp: 43-51, (1986).

[Requ80] Requicha, A.: *"Representations of Solid Objects: Theory, Methods and Systems"*. Computing Surveys of the ACM, 12 (4), pp: 437-464 (1980).

[Requ82] Requicha, A.; Voelcker, H.: *"Solid Modelling: A Historical Summary and Contemporary Assesment"*. IEEE Computer Graphics and Applications, 2 (3), pp: 9-24 (1982).

[Requ83] Requicha, A.; Voelcker, H.: *"Solid Modelling: current status and research directions"*. IEEE Computer Graphics and Applications, 3 (7), pp: 25-37 (1983).

[Requ85] Requicha, A.; Voelcker, H.: *"Boolean Operations in Solid Modelling: Boundary Evaluations and Merging Algorithms"*. Proceedings of the IEEE, vol. 73, nº 1, (1985).

[Roos97a] Rossignac, J.: *"Geometric Simplification and Compression in Multirresolution Surface Modeling"*. SIGGRAPH-97 Course Notes 25, (1997).

[Roos97b] Rossignac, J.: *"Simplification and Compression of 3D Scenes"*. EUROGRAPHICS'97 Tutorial (1997).

[Roos99] Rossignac, J.; Requicha, A.: *"Solid Modelling"*. En Webster, J. (ed.): *Encyclopedia of Electrical and Electronics Engineering*, Ed. John Wiley & Sons, (1999).

[Same84] Samet, H.: *"The quadtree and related hierarchical data structures"*. ACM Computing Surveys, 16(2), pp: 187-260 (1984).

[Same85] Samet, H.; Tamminen, M.: *"Bintrees, CSG Trees and Time"*. Computer Graphics of the ACM, 19, pp: 121-130 (1985).

[Same88a] Samet, H.; Webber, R.E.: *"Hierarchical Data Structures and Algorithms for Computer Graphics. Part I: Fundamentals"*. IEEE Computer Graphics and Applications, 8, pp: 48-68 (1988).

[Same88b] Samet, H.; Webber, R.E.: *"Hierarchical Data Structures and Algorithms for Computer Graphics. Part II: Applications"*. IEEE Computer Graphics and Applications, 8, pp: 59-75 (1988).

[Same90a] Samet, H.: *"Applications of Spatial Data Structures"*. Addison Wesley Publ., Reading, MA (1990).

[Same90b] Samet, H.: *"The Design and Analysis of Spatial Data Structures"*. Addison Wesley Publ., Reading, MA (1990).

[Shek96] Shekhar, R.; Fayyad, E.; Yagel, R.; Cornhill, J.F.: *"Octree-Based Decimation of Marching Cubes Surfaces"*. Visualization'96, pp: 335-342, San Francisco, USA (1996).

[Shir81] Shirari, S.N.: *"Representation of three-dimensional digital images"*. ACM Computing Surveys, 13(4), pp:399-423, 1981.

[Tang88] Tang, Z.; Lu, S.: *"A new algorithm for converting boundary representation to octree"*. Eurographics'88, pp: 105-116, (1988).

[Taub99] Taubin, G.: *"3D Geometric Compression and Progressive Transmission"*. EUROGRAPHIS'99 State of the Art Report, (1999).

[Thib87] Thibault, W.C.; Naylor, B.: *"Set Operations on Polyhedra Using Binary Space Partitioning Trees"*. ACM Computer Graphics, 21(4), pp: 153-162 (1987).

[Torr96] Torres, J.C.; Cano, P.; Velasco, F.; Conde, F.: *"Using octrees in non cartesian coordinate systems"*. Documento de trabajo LSI-96-1, Dpto. Lenguajes y Sistemas Informáticos, Universidad de Granada (1996).

[Vela02] Velasco, F.; Torres, J.C.; Cano, P.: *"Marching Edjes: A method for isosurface extraction"*. I Ibero-American Symposium on Computer Graphics, pp: 199-208, Guimarães, Portugal, (2002).

[Vela02b] Velasco, F.; Torres, J.C.; León, A.; *"Transmisión progresiva de Octrees de Celdas"*. Documento de trabajo LSI-02-2, Dpto. Lenguajes y Sistemas Informáticos, Universidad de Granada (2002).

[Vaně91] Vaněček, G.: *"Brep-Index: A Multidimensional Space PartitioningTree"*. International Journal of Computational Geometry and Applications, 1(3), pp: 243-262 (1991).

[Whan95] Whang, K.Y.; et al.: *"Octree-R: An Adaptative Octree for Efficient Ray Tracing"*. IEEE Transactions on Visualization and Computer Graphics, Vol. 1, nº 4, pp: 343-349, (1995).

[Yau83] Yau, M.M.; Shirari, S.N.: *"A hierarchical data structure for multidimensional images"*. Communications of the ACM, 26 (7), pp: 504-515 (1983).

UNIVERSIDAD DE GRANADA
JUNIO, 2003