Capítulo 2: Modelos de representación de objetos en 3D

2.1. Introducción

Para la realización de cualquier tipo de animación se necesita contar con un sistema que permita describir la geometría de los cuerpos que estarán involucrados en la escena. Para tal efecto se hace uso de un Sistema Geométrico, el cual es un módulo de preproceso de un sistema de diseño Asistido por Computadora (CAD) que trabaja con objetos tridimensionales [Aguilera98]. Consta generalmente de los siguientes elementos [Fig 2.1]:

Estructuras simbólicas que permiten representar objetos sólidos

Procedimientos que utilizan esas representaciones para poder responder preguntas acerca de las propiedades geométricas de los objetos representados

Interfaz de entrada para crear y editar las representaciones de los objetos y para invocar a los procedimientos

Interfaz de salida para mostrar los resultados.

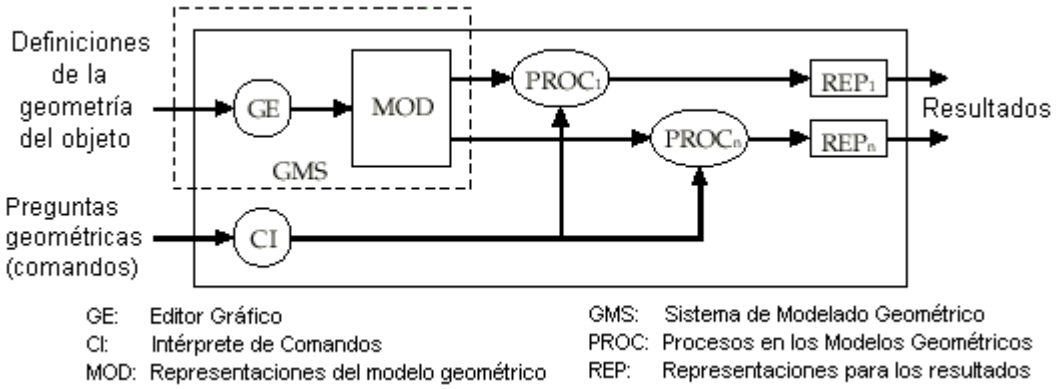

Figura 2.1: Sistema Geométrico [Aguilera98] (Traducción).

El subsistema que permite crear, visualizar y modificar las representaciones de los objetos se le llama Sistema de Modelado Geométrico (GMS).

Los algoritmos de un Sistema Geométrico no manipulan a los sólidos físicos directamente, sino a las estructuras simbólicas que los representan, y lo que diferencia a un Sistema de Modelado Geométrico de otro es, básicamente, el tipo de representación que se utiliza en el mismo [Navazo86].

En el presente capítulo se describen algunas de las más conocidas representaciones que son utilizadas actualmente en los Sistemas de Modelado Geométrico. Ya que ninguno de los esquemas existentes de representación de sólidos tiene propiedades que sean uniformemente mejores que aquellas en otros esquemas, se hará un análisis comparativo mostrando sus ventajas y desventajas individuales, así como sus capacidades para realizar operaciones booleanas, de tanto interés para el presente trabajo.

2.2.Modelo de fronteras

2.2.1. Descripción

Este modelo, también conocido como representación poliédrica de sólidos, se distingue debido a que los sólidos quedan determinados por los puntos que pertenecen a la frontera, ya que éstos separan los puntos interiores de los exteriores. La frontera es representada por un grupo de caras disjuntas que pueden ser planas o curvas [Fig 2.2]. Cuando las caras son planas, cada cara está acotada por un perímetro anular de aristas que se intersectan en vértices. Si la cara tiene agujeros, ésta queda acotada, por uno o más anillos internos de aristas. [Navazo86].

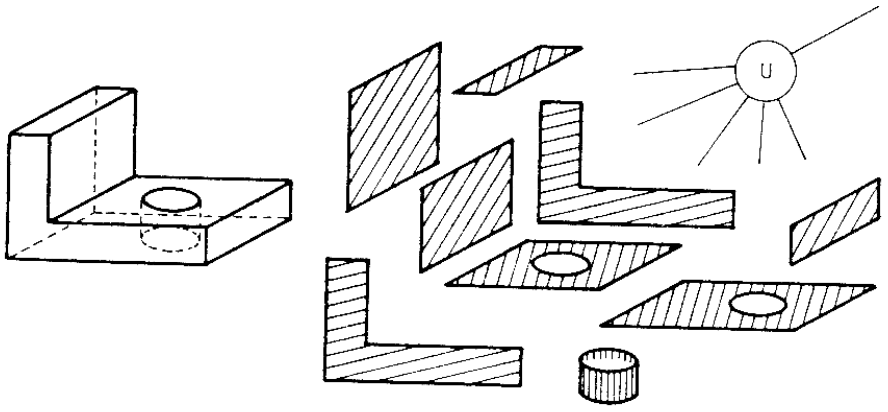

Figura 2.2: Ejemplo de un sólido y las caras que integran su frontera [Navazo86].

La información de los componentes de una superficie se divide en dos partes; una geométrica y otra topológica. La geométrica, que incluye la dimensión y localización de cada componente en el espacio; y la topológica describe la conexión entre los elementos. Es decir, mientras que la geometría define puntos, líneas y planos, la topológica identifica un punto como vértice que limita una línea que define una arista. Un anillo de aristas constituyen a un polígono como frontera de una superficie que define a una cara.

Como sea que se quiera representar la topología y la geometría de la frontera, esto debe de describir un sólido válido. Para tal efecto, es necesario cumplir las siguientes características:

- Ser cerrada. Es decir, sin caras ni aristas sueltas. Esta condición verifica si cada arista enlaza dos vértices y dos caras, y si, además el perímetro de cada cara contiene igual número de aristas que de vértices [Fig 2.3a]
- Que no haya caras que se intersecten entre si [Fig 2.3b]
- Que defina dos semiespacios, uno interior al sólido y otro exterior

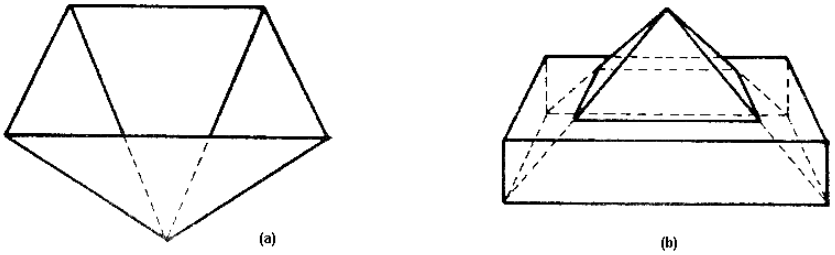

Figura 2.3: Ejemplo de sólido con superficie no cerrada (a), y con superficie que se intersecta a sí misma (b) [Navazo86].

La verificación de las dos primeras propiedades puede realizarse mediante la ecuación generalizada de Euler [Foley92]

$$C+V-A=2S+R-2H$$

donde:

C es el número de caras del objeto,

V es el número de vértices,

A es el número de aristas,

R es el número de anillos interiores de las caras

H es el número de agujeros que atraviesan al objeto

S es el número de componentes separadas (partes) del objeto

2.2.2. Representación

Este esquema es el que requiere más cantidad de información para mantener la representación. Si tomamos en cuenta que, como ya se mencionó, lo más importante, y generalmente lo más difícil también, es que los sólidos definidos mediante este esquema

sean válidos, la estructura debe facilitar al máximo el acceso a la información geométrica y topológica. Una de las estructuras más utilizadas y completa es aquella en la que se necesitan cuatro listados o vectores de información. [Fig 2.4]. El primero contiene la información geométrica de todos los vértices del objeto, es decir, las coordenadas *x*, *y*, *z* de cada punto. El segundo contiene la información topológica de las aristas del objeto, es decir, cada elemento contiene apuntadores al listado de vértices para el vértice inicial y el vértice final de cada arista del sólido. El tercero contiene la información topológica de los polígonos; cada elemento contiene n apuntadores al listado de aristas para cada una de las aristas de cada polígono del sólido, las cuales deben estar ordenadas en sentido contrario a las manecillas del reloj. Finalmente, el cuarto listado contiene la información topológica de las caras del objeto; cada elemento contiene *n* apuntadores al listado de polígonos para cada uno de los polígonos que componen cada cara del sólido, donde generalmente el primero es aquel cuya superficie es mayor, y representa el contorno externo de la cara, y los demás, si los hay, los contornos internos de la cara (agujeros) [Argüelles02].

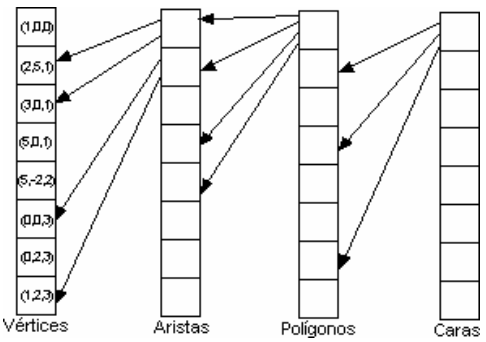

Figura 2.4: Estructura de representación del modelo de fronteras [Argüelles02]

2.2.3. Visualización

Los algoritmos de visualización requieren conocer la geometría del objeto, y es natural pensar que en este esquema resulta extremadamente sencillo de visualizar, ya

que la información esta clara y directamente representada en la estructura. De hecho, prácticamente todos los demás modelos geométricos necesitan hacer una transformación a un modelo de fronteras o un equivalente para ser visualizados. Sin embargo, la naturalidad es engañosa, ya que el paso directo del modelo a la visualización es únicamente cierto para cuando los sólidos representados son convexos y constan de un solo componente. En cualquier otro caso, el modelo de fronteras requiere de algoritmos auxiliares para la eliminación de partes ocultas; lo cual generalmente requiere de un costoso tiempo de procesamiento. [Argüelles02]

2.2.4. Operaciones Booleanas

En este modelo sucede algo interesante, y es el hecho de que la sola inclusión de una sola dimensión eleva la complejidad de los algoritmos a un nivel sencillamente impráctico. En la literatura pueden encontrarse infinidad de algoritmos para la realización de operaciones Booleanas en el modelo de fronteras cuando éste es usado para representar únicamente polígonos, es decir, objetos limitados al espacio 2D.

Cuando hablamos del caso 3D; es bastante difícil encontrar algoritmos que realicen operaciones Booleanas, aún si el dominio se reduce únicamente a los poliedros más sencillos, los convexos. Inclusive los algoritmos que operan mediante fuerza bruta, es decir, aquellos que intentan intersectar todas las caras de un sólido contra las del otro para probar las intersecciones, resultan imprácticos dada la infinidad de casos extremos, la dificultad para manejarlos y el hecho de que, si de por si, los poliedros convexos son difíciles, resulta mucho más cuando incluimos los poliedros cóncavos o el caso generalizado de los *non-manifold*. [Argüelles02]

2.3.Árboles BSP

2.3.1. Descripción

Un árbol BSP es fácil de entender si vemos su método de construcción. En [Fig 2.5] se muestra la construcción de un árbol BSP. Si tenemos una región **R** y se elige un Hiperplano **H** que intersecte a **R**, y se usa **H** para inducir una partición binaria en **R** que produzca dos nuevas regiones $\mathbf{R}^+$ y $\mathbf{R}^-$**.** Cada una de estas dos regiones resultantes puede ser a su vez recursivamente dividida, para producir un árbol binario de regiones [Naylor90].

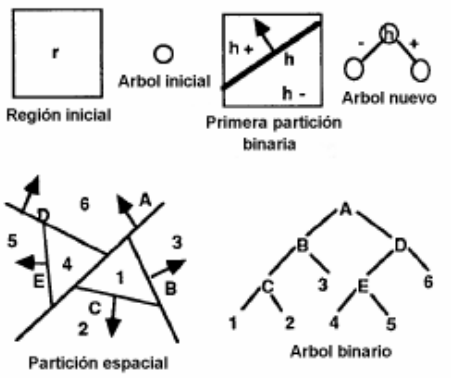

Figura 2.5: Construyendo un árbol BSP [Naylor90] (Traducción).

Así, podemos definir un árbol BSP como un conjunto jerárquico de regiones *D*-dimensionales con una relación padre-hijo definida por medio del resultado de dividir binariamente a un padre. La gráfica de estas relaciones es un árbol binario. Formalmente, un particionador binario de una *d*-región **R** es cualquier subconjunto *d-l* de **R** que divide a **R** en dos subconjuntos ajenos, $\mathbf{R}^-$ y $\mathbf{R}^+$ de tal forma que cualquier ruta entre dos puntos, $p^- \in \mathbf{R}^-$ y $p^+ \in \mathbf{R}^+$, debe intersectar al particionador binario. Aplicar esta operación recursivamente produce un árbol BSP [Naylor90]

Cada nodo interno **V** de un árbol BSP tiene un particionador binario asociado que divide a la región de **V**, mientras que cada nodo hoja corresponde a una región no particionada. Estas regiones son conocidas como celdas, las cuales pueden ser internas (*in*) o externas (*out*) al objeto a representar. Cada rama del árbol corresponde a un semiespacio: la rama izquierda al semiespacio negativo al hiperplano que particiona al padre, y la rama derecha al semiespacio positivo. Así, podemos definir una región **R** como la intersección de los semiespacios abiertos que corresponden a las ramas del árbol que forman la ruta del nodo raíz al nodo que representa la región **R**. De esta manera, si la región inicial, que típicamente es todo el espacio de la dimensión *D*, es convexa y topológicamente abierta, se deduce que todas las regiones del árbol también lo serán [Naylor90].

2.3.2. Representación

El caso de los árboles BSP, siendo éstos una estructura jerárquica con un número preestablecido y fijo de hijos, existen diversas formas de codificarlo. Particularmente resultan fácilmente adaptables las utilizadas para codificar QuadTrees/OctTrees [Navazo86] [Argüelles00]. De entre ellas, una de las más concisas es la codificación lineal en preorden transversal, conocida como codificación DF (Depth First). En la cual, para analizar un nodo determinado es necesario analizar previamente toda la descendencia de todos sus hermanos anteriores según un ordenamiento fijo entre ellos. Este ordenamiento puede ser arbitrario en el caso de los árboles binarios, siempre y cuando el ordenamiento sea congruente en todos. Así, el árbol de la figura 2.5 sería:

A(B(C(12)3)D(E(45)6))

Siendo que en un árbol BSP existen únicamente dos tipos de nodo, los internos y las celdas, es necesario 1 bit para poder distinguirlos. Además, para el caso de los nodos

internos, es necesario codificar su particionador binario asociado, lo cual se puede lograr mediante un apuntador a la ecuación orientada del hiperplano del particionador, en una tabla de ecuaciones de tales hiperplanos. Esta tabla podría codificarse representando cada plano como una tupla de cuatro coeficientes (a, b, c, d) representando los coeficientes de la forma $ax+by+cz+d=0$, para el caso 3D. Los signos de estos elementos se escogen de manera que un punto interior (x, y, z) satisfaga la desigualdad $ax+by+cz+d<0$.

2.3.3. Visualización

Los árboles BSP no escapan a la restricción antes mencionada, de que debe ser transformada a una representación de fronteras para visualizarse. Existen varios algoritmos que permiten realizar este proceso, tanto para el caso 3D [Thibault87], [Naylor90], como para el caso general n-dimensional [Baldazzi96], [Baldazzi97], [Comba96]. El procedimiento es relativamente sencillo, aunque es importante señalar que para poliedros muy complejos puede representar un tiempo de procesamiento relativamente significativo.

Una vez obtenida la representación de las fronteras de cada uno de los nodos internos del árbol BSP, el algoritmo de visualización es bastante simple. Si además consideramos que la eliminación de partes ocultas se da de manera natural, vemos que éste es uno de los modelos más poderosos y rápidos de visualización geométrica.

2.3.4. Operaciones Booleanas

En [Naylor90] se propuso un algoritmo que permitía realizar operaciones Booleanas directamente entre dos árboles BSP. Lo interesante de este algoritmo es que, si es

implementado en su totalidad, permite manejar de una manera muy efectiva los casos extremos, además de que el análisis de su complejidad demuestra que es de O($\mathbf{n^2}$), lo cual supera por mucho9 a sus equivalentes en el modelo de Fronteras, cuyo mejor caso es de O($\mathbf{n^3}$). Si a esto aunamos el hecho de que los árboles BSP son independientes del número de dimensiones de manera natural permite que estas operaciones puedan extenderse a cualquier dimensión.

2.4. OctTrees Clásicos

2.4.1. Descripción

Esta estructura de datos cuenta con mucha literatura [Samet90], [Chen88], [Elbert88], [Foley92]. Esta estructura es un árbol octal jerárquico, generado por una subdivisión recursiva de un universo cúbico finito. En esta estructura cada nodo es una hoya o tiene ocho hijos. El árbol divide al espacio del universo en cubos interiores y exteriores al objeto. La raíz del árbol representa al universo, el cual debe ser un cubo de arista 2^n. Este cubo es dividido en ocho cubos iguales de arista 2^{n-1}, denominados octantes. Cada octante es representado mediante uno de los ocho hijos ordenados de la raíz. Si un octante está parcialmente dentro del objeto, es subdividido en otros ocho cubos. Estos nuevos octantes de representarán como hijos del octante en cuestión. El proceso anterior es repetido recursivamente a fin de obtener octantes totalmente interiores o exteriores al sólido, o bien octantes con un tamaño de arista suficientemente pequeño (mínima resolución) que representa el nivel de precisión del objeto. [Argüelles02]

El tamaño y la localización de un octante quedan determinados por el nivel y por la posición de su nodo asociado dentro del árbol. A los nodos asociados a octantes que se subdividen se les denomina grises, a los asociados a octantes interiores al sólido se les llama Negros, y a los exteriores Blancos.

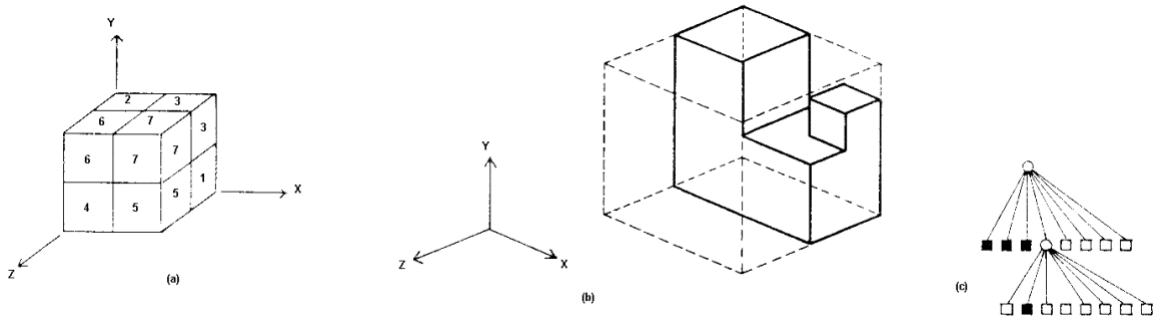

Figura 2.6: a) Un octante y su subdivisión. b) Ejemplo de un objeto simple y (c) su representación en forma de árbol octal [Navazo86].

2.4.2. Representación

De igual forma que los árboles BSP, al ser una estructura jerárquica, existen varias formas de codificar un OctTree. Algunas de ellas están descritas en [Navazo86], [Argüelles00]. Una vez más, la codificación DF resulta ser una de las más concisas, donde para analizar un nodo determinado es requisito haber previamente analizado la descendencia de los hermanos anteriores, según su ordenamiento fijo entre ellos. En el caso de los OctTrees, puede utilizarse como ordenamiento de los nodos hermanos una extensión del ordenamiento propuesto en [Samet90] para los QuadTrees. Siguiendo estamínea, si consideramos que el octante padre se encuentra localizado espacialmente como en [Figura 2.6a], es decir, cuyas coordenadas extremas con (0,0,0) y (2,2,2) respectivamente, los ocho nodos hijos quedarían ordenados de la siguiente manera (considerando su coordenada extrema inferior): (0,0,0), (1,0,0), (0,1,0), (1,1,0), (0,0,1),

(1,0,1), (0,1,1), (1,1,1) [Aguilera98]. Así pues, para la codificación del árbol [Fig 2.6c] sería:

G(NNNG(BNBBBBB)BBBB)

donde B, N y G simbolizan respectivamente los nodos Blanco, Negro y Gris. En este caso no se presentó un nodo Gris de mínima resolución, que son los nodos en los que el proceso de subdivisión se detiene al llegar a una mínima resolución preestablecida, aun cuando éste no este completamente dentro o fuera del objeto. Teniendo 4 tipos de nodos, son necesarios únicamente dos bits para su codificación. [Argüelles02].

2.4.3. Visualización

Los algoritmos de visualización de OctTrees, hacen ventaja de su estructura regular de cubos que no se intersectan entre si. Ya que un OctTree está espacialmente ordenado, es posible elegir una enumeración de los nodos de un árbol que permita visualizarlos proporcionando naturalmente una eliminación de partes ocultas. La enumeración adecuada se obtiene utilizando el punto de observación respecto al nodo padre. En [Foley92] se proporciona una manera de hacerlo: mostrando primero el octante más lejano, después los tres vecinos que tienen una cara común con dicho octante en cualquier orden, después los tres vecinos del octante más cercano en cualquier orden, y por último el octante más cercano. En [Figura 2.7] dicha enumeración para un punto de observación respecto al nodo padre en V sería 0,1,2,4,3,5,6,7. Ningún nodo en esta enumeración puede bloquear a ningún otro nodo enumerado después de él. Cuando cada octante es visualizado, sus descendientes son mostrados recursivamente en el mismo orden.

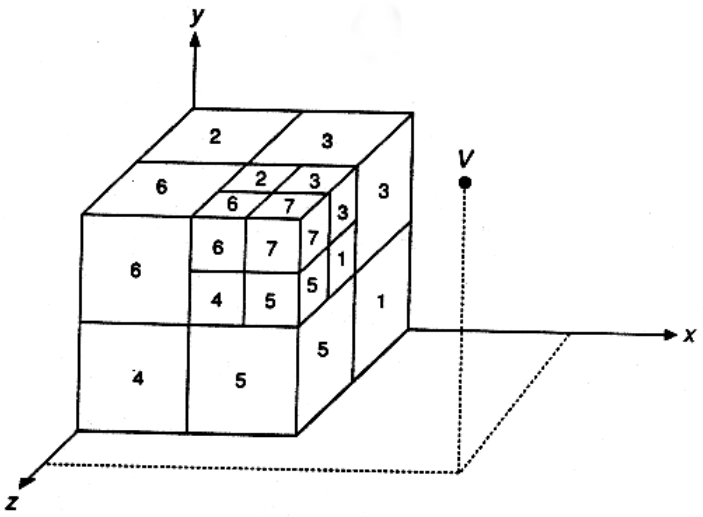

Figura 2.7: Enumeración de un OctTree para una visualización de atrás hacia adelante. (El nodo 0 está en la esquina inferior izquierda). Para un punto de bservación en V, los nodos pueden mostrarse recursivamente usando varios sistemas de ordenamiento diferentes [Foley92].

En la Tabla 2.1 se muestran ocho enumeraciones diferentes de atrás hacia delante determinadas por los signos de las tres coordenadas del punto de observación respecto al nodo padre, así como las caras del octante visibles asociadas con cada enumeración. Una coordenada x positiva o negativa del punto de observación representa que la cara derecha (R) o izquierda (L) es visible, respectivamente; de la misma manera, la coordenada y determina la visibilidad de la cara superior (U) e inferior (D) y la coordenada z la cara frontal (F) o posterior (B). Si cualquiera de las coordenadas es cero, entonces ninguna de las caras, asociadas a la coordenada correspondiente, será visible.

Tabla 2.1: Enumeraciones de atrás hacia adelante, y caras visibles [Foley92].

X	Y	Z	Enumeración	Caras Visibles
-	-	-	7, 6, 5, 3, 4, 2, 1, 0	B, D, L
-	-	+	6, 7, 4, 2, 5, 3, 0, 1	B, D, R
-	+	-	5, 4, 7, 1, 6, 0, 3, 2	B, U, L
-	+	+	4, 5, 6, 0, 7, 1, 2, 3	B, U, R
+	-	-	3, 2, 1, 7, 0, 6, 5, 4	F, D, L
+	-	+	2, 3, 0, 6, 1, 7, 4, 5	F, D, R
+	+	-	1, 0, 3, 5, 2, 4, 7, 6	F, U, L
+	+	+	0, 1, 2, 4, 3, 5, 6, 7	F, U, R

Una vez determinado el orden de pintado de los nodos, es necesario obtener la frontera de cada una de las a lo mas tres caras visibles de cada nodo a pintar. Al ir efectuando el recorrido recursivo del árbol se pueden obtener la localización y el tamaño de cada nodo si se conoce el tamaño del universo y la resolución del mismo [Navazo86], demostrando ser uno de los algoritmos de obtención de fronteras más sencillos de todos los modelos geométricos.

2.4.4. Operaciones Booleanas

Los algoritmos para realizar operaciones booleanas en este modelo son los más sencillos de entre todos los modelos de representación geométrica. Su única restricción consiste en que el universo cúbico inicial de los árboles involucrados debe ser igual tanto en tamaño como en ubicación. En caso contrario se deben ejecutar preprocesos de escalamiento y/o traslación a alguno de los árboles involucrados.

Para el caso del complemento de un OctTree, basta con recorrer la codificación del árbol cambiando los nodos Blancos por Negros y viceversa. Por lo tanto es un algoritmo de orden lineal, con una complejidad proporcional al número de nodos del árbol [Navazo86], [Argüelles00].

La intersección de dos OctTrees consta de recorrer los dos árboles comparando entre sí los nodos homólogos (mismo tamaño y localización). Según los tipos de los nodos

que se comparan se copiará al árbol de salida un tipo de nodo u otro, de acuerdo a la Tabla 2.2.

Tabla 2.2: Criterios para la creación del árbol de intersección de dos árboles a y b [Navazo86].

	B	**N**	**G**
B	**B**	**B**	**B**
N	**B**	**N**	**G***
G	**B**	**G***	**G**

B = Blanco

N = Negro

G = Gris

G* = Gris + descendencia

Hay que poner especial atención a los casos de intersección entre nodo Negro con Gris e intersección entre dos nodos Grises. En el primer caso, basta con copiar el nodo Gris con su descendencia al árbol de salida. Para el segundo caso, en contraste, se tiene que copiar un nodo Gris al árbol de salida, y seguir recursivamente con la operación de intersección entre los hijos homólogos de ambos nodos Grises. A pesar de que el algoritmo es lineal, es probable que una vez terminada la intersección deba efectuarse un recorrido del nuevo OctTree, para realizar operaciones de compactación del árbol, caso que se da cuando los 8 hermanos son del mismo tipo, ya sea Blanco o Negro. [Navazo86], [Argüelles].

Aún cuando las operaciones de unión y diferencia pueden realizarse de manera semejante a la intersección, únicamente utilizando una variación de la tabla 2.2 de acuerdo a la operación deseada, también es posible realizar éstas dos operaciones mediante el empleo de la siguiente expresión de equivalencia para el caso de la unión:

$$\mathbf{A} \cup \mathbf{B} = \varnothing(\varnothing\mathbf{A} \cap \varnothing\mathbf{B})$$

y de manera semejante para el caso de la diferencia por la expresión:

$$\mathbf{A} - \mathbf{B} = \mathbf{A} \cap \varnothing\mathbf{B}$$

2.5.OctTrees Extendidos o PM-OctTrees

2.5.1. Descripción

En los OctTrees clásicos los modelos de los objetos eran aproximados, ya que la superficie de los objetos debe representarse como un conjunto de nodos de mínima resolución, y si en el afán de lograr una mejor aproximación se reduce la escala mínima, el tamaño de almacenamiento requerido es superior. Por este motivo se propuso el modelo extendido del OctTree. Este modelo incorpora nuevos nodos terminales que contienen partes de la superficie del objeto, logrando representaciones exactas para casi todo poliedro y reduciendo el espacio de almacenamiento requerido [Navazo86], [Navazo87], [Ayala85], [Ayala 91], [Brunet85], [Argüelles00]. Los nodos terminales específicos de este modelo son:

- **Nodo Cara**: Se caracteriza por ser intersectado únicamente por una cara plana del sólido. Este nodo elimina los nodos de mínima subdivisión de todos los puntos de la superficie del objeto, restringiéndoles únicamente a las aristas del objeto
- **Nodo Arista**: Contiene únicamente partes de dos caras vecinas del sólido y parte de su arista común. Este nodo elimina los nodos de mínima subdivisión a lo largo de todas las aristas del objeto, y se limitan a los vértices del poliedro
- **Nodo Vértice**: Contiene un vértice del poliedro y partes de las caras y aristas que convergen en él. Este nodo reduce enormemente la aparición de los nodos grises de mínima subdivisión, desapareciendo prácticamente en casi todos los poliedros.

- **Nodo Casi-Vértice**: Contiene dos o más caras que convergen en un mismo vértice que se encuentra fuera del nodo. Este último nodo elimina los nodos grises de mínima subdivisión que pudiesen aún estar en el modelo.

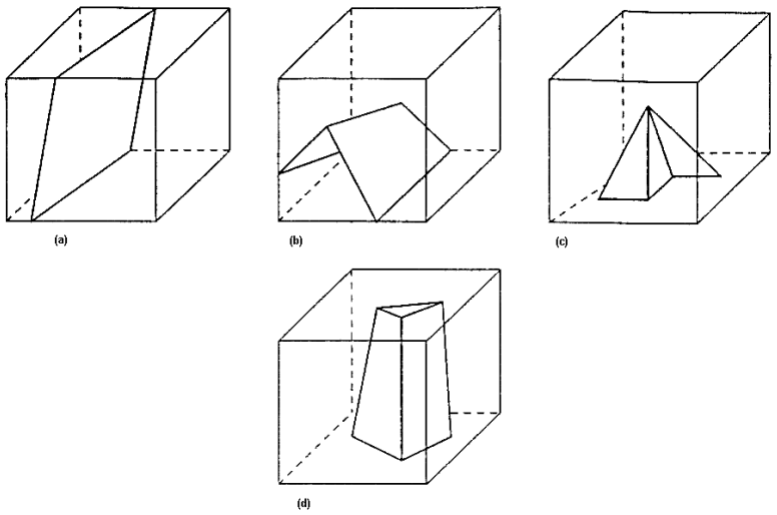

Figura 2.8: Nodos Extendidos para OctTrees: (a) Cara, (b) Arista, (c) Vértice y (d) Casi-Vértice [Ayala91].

Mientras mayor sea el número de tipos de nodos soportados en el modelo, será menor la cantidad de espacio de almacenamiento requerido para el árbol. Esto se debe al hecho de que el número total de nodos en el árbol se reduce notablemente.

2.5.2. Representación

La representación interna de un PM-OctTree mantiene dos estructuras de datos [Ayala85], [Brunet85]:

Una codificación lineal del árbol usando codificación DF. Donde para analizar un nodo determinado en esta estructura se requiere analizar previamente toda la descendencia de todos sus hermanos anteriores, según un ordenamiento fijo entre ellos.

Una tabla auxiliar que almacena las ecuaciones de soporte de las caras que definen la frontera del objeto. Cada cara de la frontera del objeto es representada por medio de cuatro números reales que corresponden a los cuatros coeficientes de la ecuación lineal $ax + by + cz + d = 0$. Los signos de la tupla (a, b, c, d) son elegidos de tal manera que los puntos (x, y, z) del interior del objeto satisfagan la desigualdad $ax + by + cz + d < 0$.

Dentro de la codificación DF del árbol, los nodos hermanos se ordenan igual que en el modelo clásico. Ya que existen ocho nodos diferentes (Negro, Blanco, Gris, Gris de mínima subdivisión, Cara, Arista, Vértice, Casi-Vértice) serán necesarios 3 bits para poder distinguirlos. La codificación de cada nodo sigue las siguientes indicaciones [Navazo86], [Argüelles00]:

- Los nodos Negros, Blancos y Grises se codifican utilizando únicamente los 3 bits identificadores
- Los nodos Cara utilizan su tipo, seguido por un apuntador a la ecuación orientada del plano asociado a la cara, en la tabla de ecuaciones
- Los nodos Arista usan su tipo, seguido de dos apuntadores a las ecuaciones orientadas de los planos asociados y un bit de configuración. Si $\prod i$ y $\prod j$ son los semiespacios internos asociados a los planos de las caras en el nodo, la configuración determina si el sólido está localizado dentro de $\prod i \cap \prod j$ (configuración 0), o si está dentro de $\prod i \cup \prod j$ (configuración 1). Es decir, el bit (o ó 1) de configuración indica si el sólido está en la parte convexa o cóncava de la arista, respectivamente
- Los nodos Vértice utilizan su tipo, seguido del número *n* de caras que convergen en el vértice, seguido de *n* apuntadores a las ecuaciones de

los planos asociados, y una lista de los *n* bits de configuración de cada una de las aristas que convergen en el vértice, las cuales están ordenadas cíclicamente.

- Los nodos casi-Vértice se codifican de manera idéntica a los nodos Vértice

2.5.3. Visualización

El modelo extendido de los OctTrees, hereda la propiedad de que su estructura regular, compuesta por cubos que no se intersectan entre sí, proporciona un ordenamiento espacial de manera natural a sus nodos, lo cual nos permite usar el mismo algoritmo para la selección del orden para presentar los nodos del árbol. Sin embargo, este algoritmo únicamente permite seleccionar una correcta visualización de los nodos del árbol, más no permite detectar el orden de pintado de las caras que están dentro del nodo. Esto no representa problema para los nodos Cara y Arista, pero en los nodos Vértice y Casi-Vértice el problema es exactamente el mismo que en el modelo de Fronteras; y esto es que el modelo en sí es incapaz de lograr una correcta eliminación de partes ocultas. Por lo cual es necesario valerse de costosos mecanismos auxiliares para obtener visualizaciones adecuadas.

En el caso de los nodos clásicos (Negro y Blanco), el proceso es exactamente igual que en el caso de un OctTree clásico. Aún para los nodos Cara y Arista, el proceso requiere únicamente de intersectar el plano de soporte de cada una de las caras asociadas al nodo con el cubo del mismo, y en el caso de los nodos Arista, además de intersectar entre sí ambos polígonos resultantes de acuerdo al bit de configuración [Navazo86], [Argüelles00]. La situación es mucho más complicada para los nodos Vértice y Casi-Vértice, puesto que además de tener que intersectar cada una de las caras

asociadas con el cubo del nodo, el proceso de intersección entre los polígonos resultantes con sus respectivos polígonos adyacentes (de acuerdo al ordenamiento cíclico) no sólo requiere de conocer los bits de configuración entre ellos, sino que además el proceso debe tomar en cuenta si los polígonos participantes forman parte de una cara convexa o cóncava del sólido, y la información almacenada en un nodo Extendido no es suficiente para obtener esta información de manera inmediata, por lo que el proceso de reconstrucción del modelo de fronteras resulta por demás complicado.

2.5.4. Operaciones Booleanas

Una vez más, los PM-OctTrees, heredan las capacidades y restricciones del modelo clásico, y esto es que ambos universos deben ser iguales en tamaño y posición; y los nodos Blanco y Negro siguen comportándose de la misma forma para las operaciones de complemento, intersección, unión y diferencia. Sin embargo, hace falta explicar el comportamiento de los nodos extendidos.

En el caso de la operación complemento, se sigue el mismo procedimiento que en un OctTree clásico, únicamente se debe considerar lo siguiente [Navazo86], [Argüelles00]:

- Deben cambiarse los signos de todos los coeficientes de todas las ecuaciones de soporte en la tabla asociada a la estructura.
- Los nodos Cara no sufren cambios.
- Los nodos Arista requieren cambiar su bit de configuración e intercambiar los dos apuntadores asociados a sus dos ecuaciones de soporte.

Sin embargo, dada la complejidad para complementar los nodos Vértices y Casi-vértice, originalmente fueron restringidos a vértices que tuvieran a lo más tres o cuatro caras incidentes, y todas las posibles configuraciones así como sus respectivas maneras de complementarse fueron listadas en [Navazo86]. Posteriormente en [Navazo89], [Ayala91], el modelo fue generalizado para soportar vértices con n caras incidentes, con la consiguiente descripción de cómo complementarlos.

Para el caso de la unión y la diferencia, pueden realizarse por medio de las fórmulas descritas anteriormente, únicamente restaría describir el comportamiento de los nodos extendidos en la operación de intersección. Antes que nada, se deben de fusionar las tablas de ecuaciones de soporte asociadas a cada uno de los árboles involucrados, posteriormente se procede a operar los árboles de manera análoga que en el caso clásico; sin embargo, cuando tenemos que operar los nodos extendidos, la operación no resulta tan sencilla. Para operar los nodos extendidos entre sí, se requiere de un procedimiento especializado que maneje cada una de las posibles combinaciones entre nodos Extendidos (C-C, C-A, C-V, A-A, A-V, V-V), y esto sin considerar a los nodos Casi-Vértice [Navazo86], [Argüelles00]. Estos procedimientos, aunque muy especializados para su uso específico, no dejan de ser variantes de algoritmos par la realización de operaciones Booleanas en el modelo de Fronteras, por lo que heredan sus principales desventajas, elevada complejidad y alto número de casos extremos). Incluso, cuando se considera el caso de la operación entre nodos Vértice o Casi-Vértice con cualquier otro tipo de nodo, los algoritmos son muy complejos e inclusive difíciles de encontrar en la literatura o ser diseñados para el caso general. Y aunque posteriormente en [Navazo89], [Ayala91] estos algoritmos fueron simplificados, aún siguieron requiriendo de varios casos especiales.

Finalmente, aún cuando se pudiesen superar estas desventajas, se sigue teniendo el problema de obtener la representación de fronteras de un nodo Vértice o Casi-Vértice; y los procedimientos especializados de intersección entre nodos extendidos requerirían de un preproceso costoso, para poder operarlos.

2.6.Análisis de modelos presentados

A continuación se muestra un análisis comparativo de los modelos presentados en el presente capítulo:

- ***Modelo de Fronteras***: A pesar de que es uno de los más usados, dada su gran facilidad de comprensión e implementación, presenta muchas desventajas, principalmente para su visualización correcta, con eliminación de partes ocultas, y la impracticidad para realizar operaciones booleanas con ellos. Si a esto le aunamos la gran cantidad de información necesaria para su creación y mantenimiento, así como los altos requerimientos de memoria, resulta sencillamente un modelo poco práctico para efectos del presente trabajo.
- ***Árboles BSP***: Es indudable que es uno de los más poderosos y populares. Su algoritmo de visualización soporta naturalmente la eliminación de partes ocultas, y las operaciones booleanas son mucho más simples que en el modelo de fronteras; además de que son muy concisos y requieren de poca información. Sin embargo, para poliedros complejos, los árboles BSP resultantes pueden llegar a ser de un gran tamaño debido a la cantidad de particiones que pueden surgir las caras, aún aplicando algoritmos de optimización [Paterson90]. De igual forma; y a diferencia al modelo OctTree, donde los algoritmos de

reconstrucción de fronteras y de operaciones Booleanas permiten operar los nodos de forma independiente entre ellos; en los árboles BSP un nodo cuya frontera sea reconstruida con algún error de precisión, puede generar un error en los demás nodos. En otras palabras, sus algoritmos no son tan delicados a los errores de precisión como el modelo de fronteras, pero para poliedros complejos, resulta muy común llegar a errores de precisión. Es así como este modelo es normalmente usado como auxiliar del modelo de fronteras, eliminando la necesidad de usar algoritmos de reconstrucción de las mismas, y utilizándolos únicamente como mecanismo de indexación del modelo de fronteras para permitir su visualización correcta. Sin embargo, esto sigue requiriendo gran cantidad de memoria.

- ***OctTrees Clásicos***: Parecerían ser uno de los modelos más efectivos y prácticos; debido a su facilidad de construcción, visualización y facilidad para realizar operaciones booleanas. Sin embargo, con este modelo no se llega a obtener representaciones exactas para ningún poliedro, además de que las estructuras obtenidas normalmente son de un tamaño considerable, por ello es que comúnmente los encontramos como estructuras auxiliares.
- ***OctTrees Extendidos o PM-OctTrees***: Resuelven las principales desventajas de los OctTrees clásicos, conservando la mayoría de sus ventajas, reduciendo en forma importante el tamaño de los árboles y aumentando la precisión de sus representaciones. Sin embargo, tienen la gran desventaja de la complejidad exagerada para realizar

operaciones booleanas con ellos, haciéndolas incluso tan complejas como en el caso del modelo de fronteras.

Tabla 2.2: Tabla comparativa de ventajas y desventajas de los modelos presentados.

	Requerimientos de Memoria	**Visualización (eliminación de partes ocultas)**	**Precisión**	**Complejidad de operaciones Booleanas**
Modelo de Fronteras	Alta	Requiere algoritmos especiales	Baja	Muy alta
Árboles BSP	Bajo	Natural	Mediana	Baja (delicado)
OctTrees	Alta	Natural	Baja	Baja
PM-OctTrees	Bajo	Natural (presenta alta complejidad para mostrar ciertos nodos)	Mediana	Muy alta