Capítulo II
Modelos de representación geométrica

2.1 Introducción

Un *Sistema Geométrico* es el módulo de preproceso de un sistema de Diseño Asistido por Computadora (CAD por sus siglas en inglés) que trabaja con objetos tridimensionales [Aguilera98]. Consta generalmente de los siguientes elementos **[Figura2.1]**:

- 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, como volumen, superficie, momento de inercia, centro de gravedad, etc.
- 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, como visualización de partes ocultas, diferentes tipos de proyecciones, etc.

Al subsistema que permite crear, visualizar y modificar las representaciones de los objetos se le llama Sistema de Modelado Geométrico (GMS por sus siglas en inglés).

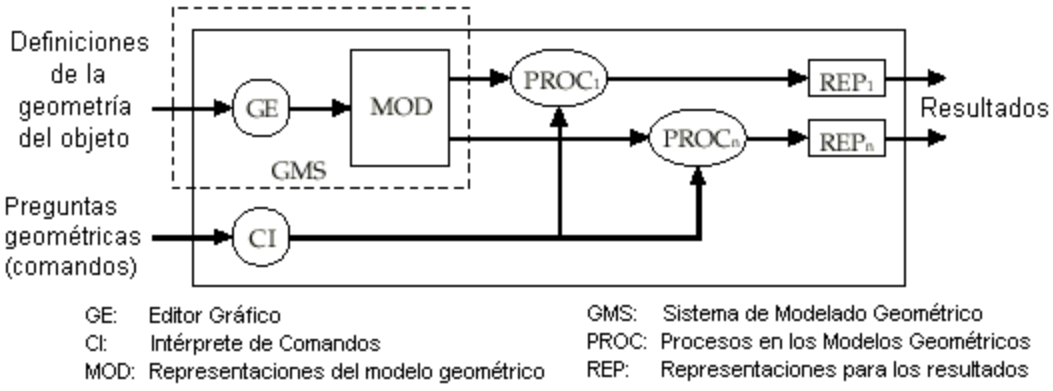

FIGURA 2.1
Sistema Geométrico [Aguilera98] (Traducción).

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].

Así pues, en este capítulo se describen algunas de las más conocidas representaciones que son utilizadas actualmente en los Sistemas de Modelado Geométrico, especialmente aquellas que se encuentran más relacionadas con el modelo propuesto por el presente trabajo. Ya que ninguno de los esquemas existentes de representación de sólidos tiene propiedades que sean uniformemente mejores que aquellas en otros esquemas, también se incluyen las principales ventajas y desventajas de dicho modelos de representación.

2.2 Modelo de fronteras

2.2.1 Descripción general

En este modelo los sólidos quedan determinados por los puntos que pertenecen a la frontera, ya que éstos separan los puntos interiores de los puntos exteriores del sólido. La frontera es representada por un conjunto disjunto de caras que pueden ser planas o curvas **[Figura2.2]**. 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, queda acotada, a su vez, por uno o más anillos internos de aristas. A este tipo de representación se le llama, también, representación poliédrica de sólidos [Navazo86].

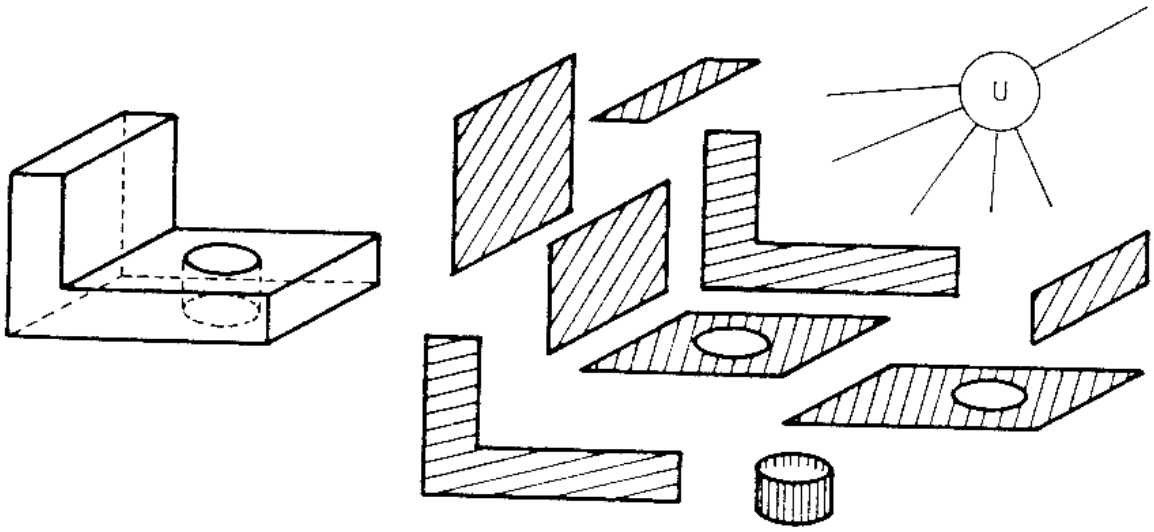

FIGURA 2.2
Ejemplo de un sólido y las caras que integran su frontera [Navazo86].

La información asociada a los componentes de una superficie (caras, aristas, vértices) comprende dos partes. Una es la geométrica, que incluye la dimensión y localización en el espacio de cada componente. La otra es la topológica, que describe la conexión entre los elementos. Así, la geometría define puntos, líneas y planos, y la topología 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.

Cualquiera que sea la representación escogida para la topología y para la geometría de la frontera, lo más importante es que esté definido un sólido válido. Para conseguir que la frontera defina un sólido válido, las características que la superficie del sólido debe cumplir son las siguientes:

- Ser cerrada. Es decir, sin caras ni aristas sueltas. Esta condición se 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 **[Figura2.3a]**.
- Que no haya caras que se intersecten entre sí **[Figura2.3b]**.
- Ser orientable. Es decir, que defina dos semiespacios, uno interior al sólido y otro exterior.

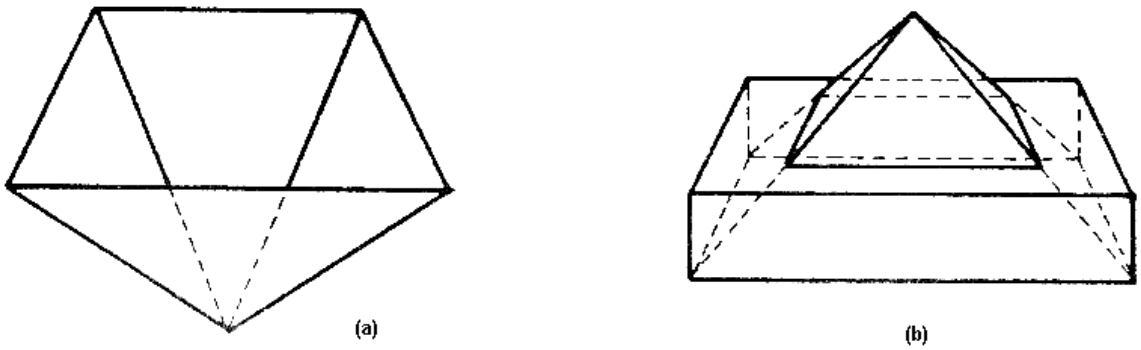

FIGURA 2.3
Ejemplo de sólido con superficie no cerrada (a),
y con superficie que se intersecta a sí misma (b) [Navazo86].

La verificación de las dos primeras propiedades puede realizarse mediante la ecuación generalizada de Eüler [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, y S es el número de componentes separadas (partes) del objeto.

2.2.2 Representación

Este esquema es prácticamente el que requiere más cantidad de información para mantener la representación. Dado 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 tanto geométrica como topológica. De esta manera, una de las estructuras más utilizadas y completa es aquella en la que se necesitan cuatro listados o vectores de información

[Figura2.4]. El primero de ellos contiene directamente 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 dos 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 del objeto, es decir, cada elemento contiene *n* apuntadores al listado de aristas para cada una de las aristas de cada polígono del sólido, las cuales generalmente están ordenadas en sentido contrario a las manecillas del reloj. Por último, se tiene un listado que contiene la información topológica de las caras del objeto, es decir, cada elemento contiene *n* apuntadores al listado de polígonos para cada uno de los polígonos que componen cada cara del sólido, de los cuales 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 (agujeros) de la cara.

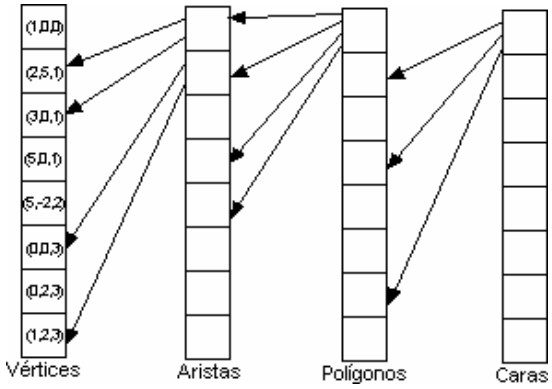

FIGURA 2.4
Estructura de representación del modelo de fronteras (elaboración propia).

2.2.3 Visualización

Ya que los algoritmos de visualización requieren conocer la geometría del objeto, a simple vista este esquema resulta extremadamente sencillo de visualizar, ya que esta información está representada directamente en su estructura. De hecho, prácticamente todos los demás esquemas de representación necesitan proveer algún tipo de algoritmo de conversión al modelo de fronteras o equivalente para poder ser visualizados. Sin embargo, esto es únicamente cierto para cuando los sólidos representados mediante este esquema son convexos y constan de una sola componente. En cualquier otro caso, el modelo de fronteras es incapaz, por sí mismo, de proveer mecanismos adecuados para la eliminación de partes ocultas, por lo que es necesario convertir el modelo a otro esquema de representación para obtener la información necesaria, o bien utilizar algoritmos que operan directamente sobre los píxeles (el espacio imagen), lo cual generalmente requiere de un costoso tiempo de procesamiento.

2.2.4 Operaciones Booleanas

Este es uno de los casos más interesantes donde la inclusión de una sola dimensión eleva la complejidad de los algoritmos a un nivel sencillamente impráctico. En otras palabras, investigando un poco la literatura al respecto pueden encontrarse infinidad de algoritmos, algunos de ellos extremadamente eficientes y elegantes, para la realización de operaciones Booleanas en el modelo de fronteras cuando éste es utilizado para representar únicamente polígonos, es decir, objetos limitados al espacio 2D. Algunos de estos

algoritmos consideran únicamente polígonos convexos [Toussaint85], dado que son mucho más sencillos de operar que los polígonos cóncavos. Sin embargo, esto no ha impedido que se hayan desarrollado algoritmos que operen no sólo con polígonos cóncavos también, sino con polígonos generales que incluyen a los *non-manifold* [Greiner98], [Maillot92], [Vatti92]. Y, aunque el manejo de los casos extremos no es siempre 100% efectivo en todos estos algoritmos, muchos de ellos incluyen mecanismos para evitarlos.

Ahora bien, hablar del caso 3D es totalmente otro tema. Es bastante difícil encontrar algoritmos que realicen operaciones Booleanas en estos casos, inclusive 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 de otro para probar las intersecciones de los polígonos, clasificar las fronteras de un sólido respecto al otro, y, según la operación a realizar, conservar unas caras u otras, resultan imprácticos dada la inmensa cantidad de casos extremos, la dificultad para manejarlos, y el hecho de que si ya el caso de poliedros convexos es bastante difícil, lo es mucho más aún cuando se incluyen a los poliedros cóncavos o, peor aún, para el caso generalizado que incluye a los *non-manifold*. Dada esta situación, se puede encontrar literatura enfocada al tema de descomponer poliedros cóncavos en convexos [Schewchuk99], [Edelsbrunner95], ya que como se mencionó, aún cuando manipular poliedros convexos tiene sus complicaciones, éstas no son comparables con las que surgen al intentar manipular directamente poliedros cóncavos.

2.3 Arboles BSP

2.3.1 Descripción general

La manera más sencilla de comprender a los árboles BSP es a través del proceso que permite construirlos. En **[Figura2.5]** se muestra la construcción de un árbol BSP. Se empieza con una región R del espacio, se elige un hiperplano H que intersecte a R, y entonces se usa H para inducir una partición binaria en R que produzca dos nuevas regiones: R^+ y R^-. Cada una de estas dos regiones resultantes puede ser a su vez recursivamente dividida, para producir un árbol binario de regiones [Naylor90].

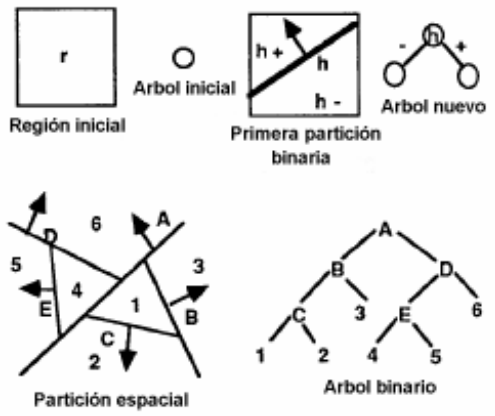

FIGURA 2.5
Construyendo un árbol BSP [Naylor90] (Traducción).

Luego entonces, un árbol BSP es un conjunto jerárquico de regiones D-dimensionales con una relación padre-hijo definida por medio de "los hijos resultantes de dividir binariamente a un padre". La gráfica de estas relaciones es un árbol binario. Más formalmente, un particionador binario de una d-región R es cualquier subconjunto d-1 de R

que divide a R en dos subconjuntos ajenos, R^- y R^+, de tal forma que cualquier ruta entre dos puntos, $p^- \in R^-$ y $p^+ \in R^+$, debe intersectar al particionador binario. Aplicar esta operación recursivamente produce un árbol BSP [Naylor90].

En un árbol BSP, cada nodo interno V 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 denominadas celdas, las que pueden ser clasificadas como celdas internas (*in*) o externas (*out*) al objeto a representar. Cada rama del árbol binario corresponde a un semiespacio: la rama izquierda al semiespacio negativo del hiperplano del nodo padre, y la rama derecha al semiespacio positivo. Luego entonces, puede definirse 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 son [Naylor90].

2.3.2 Representación

Existen diversos esquemas que permiten codificar estructuras jerárquicas de árboles, especialmente si éstos tienen un número preestablecido y fijo de hijos para cada nodo padre. Este es precisamente el caso de los árboles BSP, que son árboles binarios, por lo que resulta casi inmediato el poder utilizar alguna de estas estructuras para codificarlos. 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 preorder transversal, también conocida como codificación-DF (*Depth First* por sus siglas en inglés). 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. En el caso de un árbol BSP, puede elegirse arbitrariamente el ordenamiento de los nodos hermanos en la estructura (es decir, R^- seguido de R^+ o viceversa), siempre y cuando el ordenamiento sea el mismo para todos. De esta manera, una codificación del árbol en **[Figura2.5]** podría ser:

A (B (C (1 2) 3) D (E (4 5) 6))

En este ejemplo, y para una mejor comprensión, se ha agregado la parentización de los nodos hijos obtenidos a partir de las divisiones recursivas de cada región.

Como ya se mencionó, en un árbol BSP existen básicamente dos tipos de nodo: los internos y las celdas, por lo que únicamente es necesario 1 bit para poder distinguirlos. Además, para el caso de los nodos internos, es necesario codificar también su particionador binario asociado, lo que puede hacerse mediante un apuntador a la ecuación orientada del hiperplano del particionador, en una tabla de ecuaciones de dichos hiperplanos (de una manera equivalente a como se codifican los nodos Arista/Cara en los QuadTrees/OctTrees Extendidos [Navazo86], [Argüelles00]). En **[Sección2.5.2]** se detalla con más profundidad una manera para codificar una tabla de ecuaciones de soporte, que en este caso particular serían las ecuaciones de los hiperplanos de los particionadores binarios.

2.3.3 Visualización

Como ya se mencionó en **[Sección2.2.3]**, la mayoría de los esquemas de representación requieren convertirse al modelo de fronteras o a alguno equivalente para poder ser visualizados. Los árboles BSP no son la excepción. En la literatura podemos encontrar una buena cantidad de algoritmos que permiten realizar este proceso, tanto para el caso 3D [Thibault87], [Naylor90], como para el caso general n-dimensional [Baldazzi96], [Baldazzi97], [Comba96]. En **[Sección4.2.1]** se detalla con más profundidad el algoritmo descrito en [Naylor90], el cual permite obtener la representación de las fronteras de cualquier objeto sólido definido mediante árboles BSP. El procedimiento es relativamente sencillo, aunque es importante señalar que para poliedros muy complejos puede representar un tiempo de procesamiento relativamente significativo

Ahora bien, una vez obtenida la representación de las fronteras de cada uno de los nodos internos de un árbol BSP, el algoritmo que permite visualizarlo es bastante simple, y si se considera que además proporciona eliminación de partes ocultas de manera natural, se convierte en uno de los algoritmos más poderosos y rápidos de visualización geométrica. Este algoritmo es detallado con más profundidad en **[Sección4.2]**.

2.3.4 Operaciones Booleanas

El hecho de que los árboles BSP sean independientes del número de dimensiones de manera natural permite que la realización de operaciones Booleanas entres ellos sea

relativamente fácil de extender a cualquier dimensión. En [Thibault87] se propuso un algoritmo para realizar operaciones Booleanas entre dos sólidos, uno definido mediante un árbol BSP y el otro de ellos definido mediante un modelo de Fronteras, que desde luego estaba limitado por la ya mencionada dificultad de extender el modelo de Fronteras a otras dimensiones. Sin embargo, en [Naylor90] se propuso un algoritmo que finalmente permitía realizar operaciones Booleanas directamente entre dos árboles BSP. Lo más 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 $O(n^2)$, lo cual supera por mucho a sus equivalentes en el modelo de Fronteras, que en el mejor caso serían de orden $O(n^3)$. Este algoritmo es descrito con más detalle en **[Sección5.2]**.

2.4 OctTrees clásicos

2.4.1 Descripción general

Existen mucha literatura acerca de esta estructura de datos [Samet90], [Chen88], [Elber88], [Foley92]. Consta de una estructura de árbol octal jerárquico generada por una subdivisión recursiva de un universo cúbico finito. En esta estructura cada nodo es una hoja 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 (un cubo de arista 2^n). Este cubo es dividido en ocho cubos iguales de aristas 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 subdivido en otros ocho cubos. Estos nuevos octantes se representarán como hijos del octante en cuestión **[Figura2.6]**. 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 (denominado resolución mínima) que representan el nivel de precisión del objeto.

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 asociados a octantes exteriores se les llama Blancos.

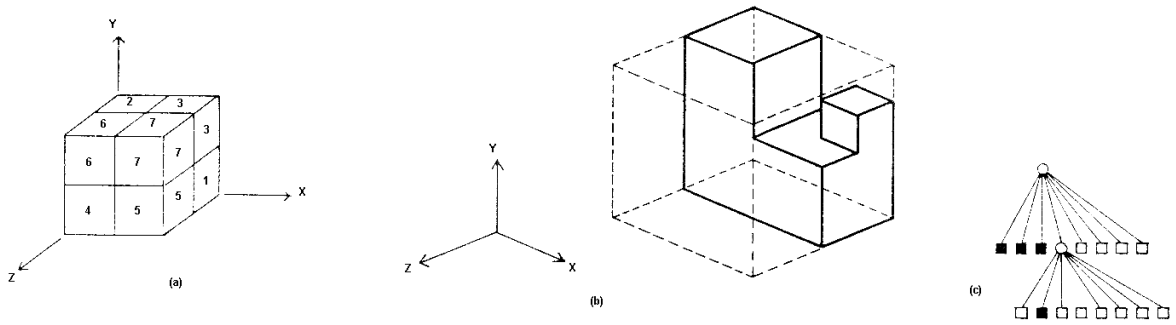

FIGURA 2.6
a) Un octante y su subdivisión.
b) Ejemplo de un objeto simple y (c) su representación en forma de árbol octal [Navazo86].

2.4.2 Representación

Como ya se mencionó en **[Sección2.3.2]**, hay varios esquemas para codificar estructuras jerárquicas de árboles. Algunos de ellos están descritos en [Navazo86],

[Argüelles00]. La codificación-DF resulta ser una de las más concisas, siendo una estructura que para analizar un nodo determinado requiere analizar previamente toda la descendencia de todos sus hermanos anteriores, según un ordenamiento fijo entre ellos. Para el caso de los OctTrees, dado que éstos son árboles octales, puede utilizarse como ordenamiento de los nodos hermanos a una extensión del ordenamiento propuesto en [Samet90] para los QuadTrees. Siguiendo esta línea, si consideramos que el octante padre se encuentra localizado espacialmente como en **[Figura2.6a]**, es decir, cuyas coordenadas extremas son (0,0,0) y (1,1,1) 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 en **[Figura2.6c]** sería:

G (N N N G (B N B B B B B B) B B B B)

donde B, N y G simbolizan, respectivamente, los tipos de nodo Blanco, Negro y Gris. Nuevamente, para una mejor comprensión, se ha agregado la parentización de los subárboles que son descendencia de los nodos grises. Además, se han omitido también los nodos Grises de mínima resolución, es decir, aquellos nodos en los que el proceso de subdivisión se detuvo por llegar a la mínima resolución preestablecida, aún cuando estos no se encuentren aún totalmente en el interior o exterior del objeto. Como existen cuatro tipos de nodos, serán necesarios únicamente dos bits para codificar cada uno.

2.4.3 Visualización

Los algoritmos para visualizar objetos codificados mediante OctTrees toman ventaja de su estructura regular, compuesta por cubos que no se intersectan entre sí. Ya que un OctTree está espacialmente ordenado, es posible elegir una enumeración de los nodos del árbol que permita visualizarlos, proporcionando además eliminación de partes ocultas. Una enumeración de atrás hacia adelante puede ser determinada utilizando simplemente el punto de observación respecto al nodo padre. En [Foley92] se proporciona una manera de hacerlo: mostrar primero el octante más lejano, después aquellos 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 **[Figura2.7]**, 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. A medida que cada octante es visualizado en pantalla, sus descendientes son mostrados recursivamente en este orden. Además, ya que cada nodo hoja es un cubo, a lo más tres de sus caras son visibles, las cuales también pueden ser calculadas directamente a partir del punto de observación respecto al nodo padre.

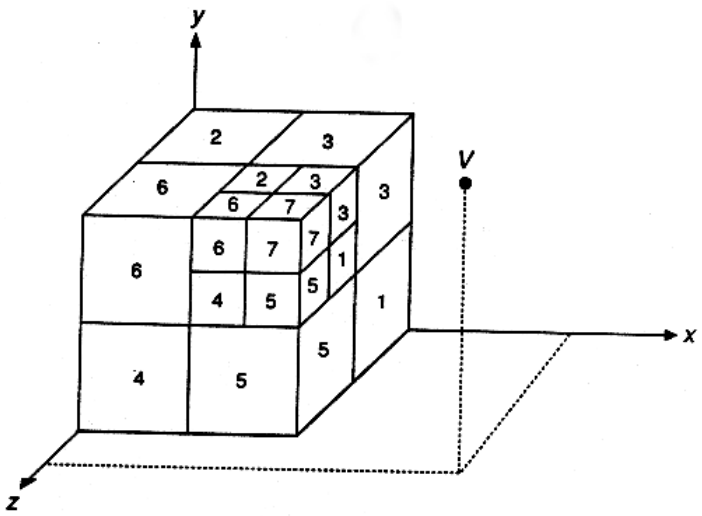

FIGURA 2.7
Enumeración de un OctTree para una visualización de atrás hacia adelante. (El nodo 0 está en la esquina inferior izquierda). Para un punto de observación en *V*, los nodos pueden mostrarse recursivamente usando varios sistemas de ordenamiento diferentes [Foley92].

En **[Tabla2.1]** se muestran ocho enumeraciones diferentes de atrás hacia adelante determinadas por los signos de las tres coordenadas del punto de observación respecto al nodo padre (que se calcula simplemente restando las coordenadas (*x*, *y*, *z*) del punto de observación menos las coordenadas (*x*, *y*, *z*) del centro del nodo padre), así como las caras del octante visibles asociadas con cada enumeración. Una coordenada *x* positiva o negativa del punto de observación respecto al nodo padre 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* controla la visibilidad de las caras frontal (F) y posterior (B). Si cualquiera de las coordenadas es cero, entonces ninguna de las caras asociadas es visible. Solo las coordenadas diferentes a cero son significativas al determinar una enumeración. Ya que todos los nodos de un OctTree están orientados idénticamente, las caras visibles y las proyecciones poligonales relativas para todos los nodos hijos de un mismo padre necesitan calcularse una sola vez si se utiliza

proyección paralela. Además, si en lugar de utilizar proyección paralela se utiliza la perspectiva, todavía se tiene la ventaja de que ya que gran parte de las caras de los nodos hijos de un mismo padre no son sino fragmentos de las caras del nodo padre, la visibilidad de dichas caras puede calcularse una sola vez utilizando únicamente al nodo padre, por lo que únicamente se necesitaría recalcular para las caras internas de los nodos hijos que no forman parte de la frontera de su padre.

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

Ahora bien, una vez determinado el orden de pintado de los nodos de un OctTree, es necesario obtener la frontera de cada una de las a lo más tres caras visibles de cada nodo a pintar. Sin embargo, al ir efectuando el recorrido recursivo por el árbol se pueden obtener la localización y el tamaño de cada nodo si se conocen el tamaño del universo y la resolución del mismo [Navazo86], con lo cual el problema queda resuelto de una manera muy simple, demostrando ser uno de los algoritmos para la obtención de fronteras más sencillos de todos los modelos geométricos.

2.4.4 Operaciones Booleanas

Este modelo presenta algunos de los algoritmos más sencillos para la realización de operaciones Booleanas entre todos los modelos de representación geométrica. Su única restricción consiste en que el universo cúbico inicial de los árboles a operar debe ser del mismo tamaño y ubicación. En caso contrario, deben aplicarse preprocesos de escalamiento y/o translación a alguno de los dos árboles.

Así pues, para la realización del complemento de un OctTree consiste en recorrer la codificación del árbol cambiando la codificación de 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].

Por otro lado, la realización de la intersección de dos OctTrees consiste en 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 lo indicado en **[Tabla2.2]**.

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

Merecen especial mención los casos de intersección entre nodo Negro con Gris e intersección entre dos nodos Grises. Para el primero, el resultado consiste en copiar al árbol de salida el nodo Gris y toda su descendencia. Para el segundo, el resultado consiste en copiar al árbol de salida un nodo Gris y luego continuar recursivamente con la operación de intersección entre los hijos homólogos de ambos nodos Grises. El algoritmo presentado es lineal, ya que se basa en un recorrido lineal de cada uno de los árboles a intersectar. No obstante, es posible que una vez terminada la intersección deba efectuarse un recorrido del OctTree creado, para realizar una compactación del árbol si se han generado ocho hermanos terminales del mismo tipo [Navazo86], [Argüelles00].

Por último, aún cuando las operaciones de unión y diferencia pueden realizarse de manera muy semejante a la intersección, únicamente utilizando una variación de **[Tabla2.2]** de acuerdo a la operación deseada, también es posible realizar estas dos operaciones mediante el empleo de la siguiente expresión equivalente para el caso de la unión:

(F2.1) $$A \cup B = \neg(\neg A \cap \neg B)$$

y, de manera semejante, con la siguiente expresión para el caso de la diferencia:

(F2.2) $$A - B = A \cap \neg B$$

2.5 OctTrees Extendidos o PM-OctTrees

2.5.1 Descripción general

Como ya se mencionó, los OctTrees clásicos son modelos sólidos que son generados mediante la división recursiva de un universo cúbico finito, y representan el árbol de este proceso de subdivisión. Los nodos terminales válidos son Blancos (completamente fuera del sólido), y Negros (completamente dentro del sólido). Siempre que un octante no puede ser representado como un nodo terminal válido, se le denomina nodo Gris y es subdividido en ocho cubos idénticos. Este proceso se repite recursivamente hasta que se llega a nodos terminales o a cubos de una mínima escala preestablecida. Son modelos aproximados, ya que la superficie de los objetos debe representarse como un conjunto de nodos de mínima subdivisión. Además, si la resolución mínima es reducida para aumentar la precisión, un espacio superior de almacenamiento es requerido.

El modelo de OctTrees Extendidos 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], [Navazo89], [Ayala85], [Ayala91], [Brunet85], [Argüelles00]. Los nodos terminales específicos de este modelo son **[Figura2.8]**:

- Nodo Cara: Se caracteriza por ser intersectado únicamente por una cara plana del sólido. Al agregar este tipo de nodo al modelo, los nodos de mínima subdivisión ya no

aparecen en todos los puntos de la superficie del objeto, sino que únicamente aparecen a lo largo de las aristas del objeto.

- Nodo Arista: Contiene únicamente partes de dos caras vecinas del sólido y parte de su arista común. Al agregar este tipo de nodo al modelo, los nodos de mínima subdivisión ya no aparecen a lo largo de todas las aristas del objeto, sino que se obtienen únicamente en las cercanías de los vértices del poliedro.
- Nodo Vértice: Contiene un vértice del poliedro y partes de las caras y aristas que convergen en él. Al agregar este tipo de nodo al modelo, los nodos de mínima subdivisión se reducen notablemente, 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. Al agregar este tipo de nodo al modelo, **algunos** de los nodos de mínima subdivisión que podrían permanecer en el modelo desaparecen.

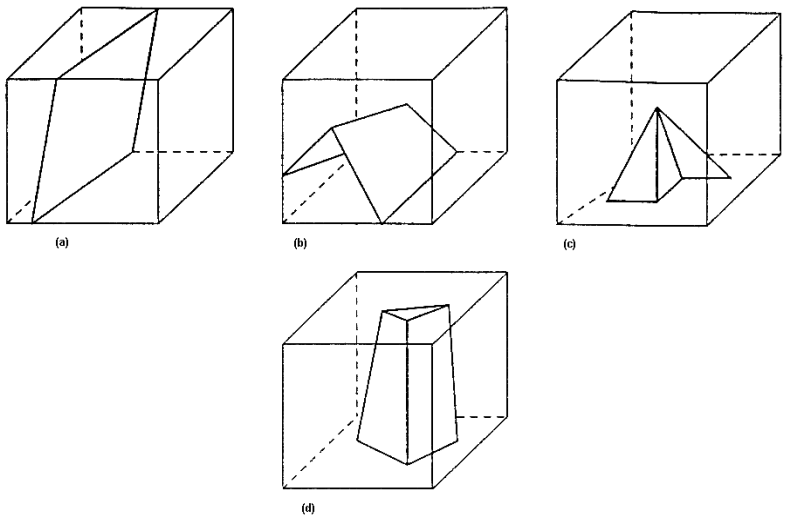

FIGURA 2.8
Nodos Extendidos para OctTrees: (a) Cara, (b) Arista, (c) Vértice y (d) Casi-Vértice [Ayala91].

Entre 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 donde la información se almacena usando codificación-DF **[Sección2.3.2]**, **[Sección2.4.2]**.
- 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 cuatro 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 como se mencionó en **[Sección2.4.2]**, es decir, igual que en el modelo clásico. Ya que existen ocho diferentes tipos de nodo (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 utilizan su tipo, seguido de dos apuntadores a las ecuaciones orientadas de los planos asociados y un bit de configuración. Si P_i y P_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 $P_i \cap P_j$ (configuración 0), o si está dentro de $P_i \cup P_j$ (configuración 1). En otras palabras, el bit (0 ó 1) de configuración indica si el sólido está en la parte convexa o en la parte 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

Los OctTrees Extendidos heredan la propiedad de los OctTrees clásicos de que su estructura regular, compuesta por cubos que no se intersectan entre sí, proporciona un ordenamiento espacial de manera natural a sus nodos, por lo que pueden utilizar el mismo algoritmo que permite elegir una enumeración de los nodos del árbol para visualizarlos,

proporcionando al mismo tiempo eliminación de partes ocultas. Sin embargo, el algoritmo de visualización del modelo clásico únicamente permite detectar el orden en que los nodos deben pintarse, pero no es suficiente para determinar el orden de pintado de las caras internas de los nodos Extendidos. Esto no representa mayor problema para los nodos Cara y Arista, pero en el caso de los nodos Vértice y Casi-Vértice el problema es exactamente el mismo que en el modelo de Fronteras: el modelo es incapaz, por sí mismo, de proveer mecanismos adecuados para la eliminación de las partes ocultas de las caras internas de los nodos Vértice y Casi-Vértice. Nuevamente, es necesario valerse de costosos mecanismos auxiliares para obtener visualizaciones adecuadas.

Por otro lado, como ya se ha manejado repetidas veces, es necesario utilizar algún mecanismo para obtener la representación en Fronteras o alguna semejante de cualquier modelo geométrico con la finalidad de poder visualizarlo. En el caso de los nodos clásicos en un OctTree Extendido, el proceso es tan trivial como el mencionando en **[Sección2.4.3]**. 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 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 caras convexas o cóncavas del sólido, y la información almacenada en un nodo Extendido no es suficiente

para obtener de manera inmediata esta información, por lo que el proceso de reconstruir el modelo de fronteras de estos nodos resulta por demás complicado, o inclusive imposible.

2.5.4 Operaciones Booleanas

Nuevamente, los PM-OctTrees tienen la interesante característica de poder heredar gran parte de las ventajas de los OctTrees clásicos en esta área. Aún cuando siguen conservando la restricción de que el universo cúbico inicial de los árboles a operar debe ser del mismo tamaño, pues requieren preprocesos de escalamiento en caso contrario, los nodos clásicos de un PM-OctTree pueden seguirse operando de manera idéntica que en un OctTree clásico, tanto en la operaciones de complemento, como en las de intersección, unión y diferencia. En otras palabras, sólo es necesario describir el comportamiento de los nodos Extendidos.

De esta manera, para complementar un OctTree Extendido debe realizarse lo mismo que en un OctTree clásico, únicamente considerando 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, para complementar los nodos Vértice y Casi-Vértice, dada su elevada dificultad, 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 cualquier número de caras incidentes, con la consiguiente descripción de cómo complementarlos. Sin embargo, la dificultad técnica de este procedimiento excede los alcances de este trabajo y se deja simplemente como consulta opcional.

Por otro lado, ya que las operaciones de unión y diferencia pueden realizarse por medio de las fórmulas descritas en **[Sección2.4.4]**, únicamente restaría describir el comportamiento de los nodos Extendidos en la operación de intersección, puesto que los nodos clásicos se comportan igual que en el modelo clásico, y las tablas de ecuaciones de soporte asociadas a cada uno de los dos árboles requieren únicamente de fusionarse en una misma. Sin embargo, esto no resulta tan sencillo en este modelo, llegando a situaciones casi tan complejas como las descritas en **[Sección2.2.4]** para el modelo de Fronteras. En otras palabras, para operar entre sí a los nodos Extendidos 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 para la realización de operaciones Booleanas en el modelo de Fronteras, por lo que heredan las principales desventajas de esta familia de algoritmos (elevada complejidad y alto número de casos extremos). Además, cuando se consideran los casos de operación entre nodos Vértice o Casi-Vértice con

cualquier otro tipo de nodo, los algoritmos necesarios resultan ser prácticamente tan complejos y difíciles de encontrar en la literatura o de ser diseñados como los algoritmos para el caso general. Peor aún, si un nodo Vértice o Casi-Vértice se opera con otro nodo Extendido de menor tamaño, pueden obtenerse configuraciones de caras que no es posible adaptar a ninguno de los procedimientos especializados considerados para las combinaciones mencionadas anteriormente, por lo que más casos especiales aparecen. Y, aunque posteriormente en [Navazo89], [Ayala91] estos algoritmos fueron simplificados, aún siguieron requiriendo de varios casos especiales.

Por último, aún cuando todas las desventajas anteriores fueran superadas, el problema descrito en **[Sección2.5.3]** para obtener la representación de fronteras de un nodo Vértice o Casi-Vértice aún persiste. Y los procedimientos especializados de intersección entre nodos Extendidos requieren que previamente se obtenga dicha representación para ambos nodos involucrados.

2.6 Algunos otros modelos

Los modelos presentados a lo largo de este capítulo no son los únicos existentes. Sin embargo, sí representan los modelos más utilizados y relacionados con el modelo propuesto por el presente trabajo. A continuación se describen de manera muy breve algunos otros modelos importantes o relacionados.

2.6.1 Arboles CSG

La Geometría Constructiva de Sólidos (CSG) es un esquema donde sólidos primitivos simples son combinados mediante operadores Booleanos regularizados que están incluidos directamente en la representación. Un objeto es almacenado como un árbol ordenado con operadores en los nodos internos, y primitivas simples en las hojas [Navazo86]. Algunos nodos representan operadores Booleanos, mientras otros realizan traslaciones, rotaciones o escalamientos **[Figura2.9]**.

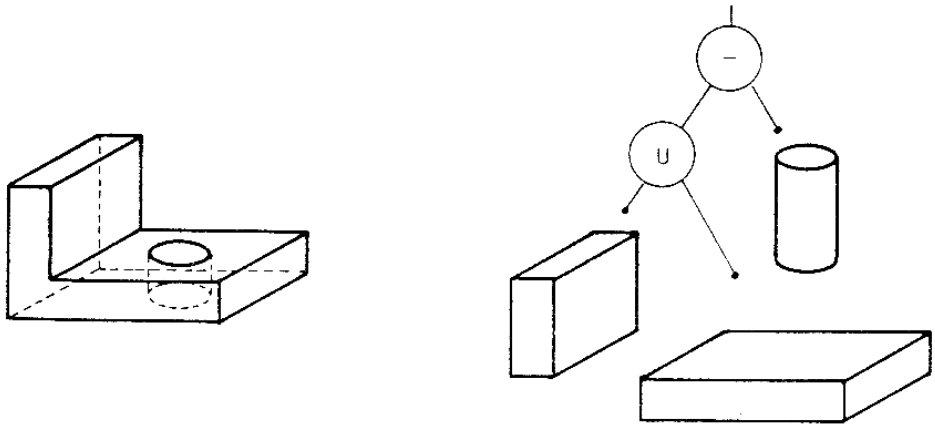

FIGURA 2.9
Ejemplo de un sólido y del árbol CSG con las operaciones necesarias para construirlo [Navazo86].

Los árboles CSG son concisos, no ambiguos y cerrados, pero no únicos. Son fáciles de crear y editar. Su dominio depende de las primitivas disponibles, así como del conjunto de operadores soportados. Sólo se necesitan simples verificaciones sintácticas locales para validar un árbol CSG (que siempre está limitado, si sus primitivas lo están) [Aguilera98].

Si bien la realización de operaciones Booleanas entre dos sólidos es sencilla, ya que consiste en combinar los dos árboles para producir el árbol final, la visualización del modelo y la realización de cálculos geométricos son complejas [Navazo86].

2.6.2 SP-OctTrees

Son una extensión a los OctTrees clásicos. Proponen la incorporación de información de la frontera del sólido no únicamente en los nodos terminales, sino también en los nodos internos del árbol. De esta manera, no siempre es necesario tener que descender al nivel más bajo del árbol para acceder a esa información, y con ello es posible acelerar las operaciones básicas del modelo [Cano02]. Además, mientras que en los OctTrees Extendidos la información de la frontera del sólido se incluye únicamente en los nodos terminales produciendo que varios de estos nodos vecinos compartan las mismas caras fronterizas, en este modelo la información no aparece repetida en los nodos terminales vecinos que comparten dichas caras, haciéndolo más conciso.

Este modelo propone utilizar cuatro tipos de nodos:

- Nodos Blancos: Equivalentes a los del modelo clásico.
- Nodos Convexos: Cuando la intersección entre un nodo y el sólido a representar es convexa se utilizan estos nodos, incluyendo apuntadores a las ecuaciones de los planos que se encuentran en el *convex hull* del sólido representado en dicho nodo.

- Nodos Cóncavos: Cuando la intersección entre un nodo y el sólido a representar es cóncava se utilizan estos nodos, incluyendo también apuntadores a las ecuaciones de los planos que se encuentran en el *convex hull* del sólido representado en dicho nodo. Las normales de estos planos deben invertirse cuando desea manipularse este nodo en procesos posteriores.
- Nodos Grises: Cuando se encuentran tanto convexidades como concavidades en la intersección entre un nodo y el sólido a representar se utilizan estos nodos, dividiéndolos recursivamente como en el modelo clásico, pero incluyendo en ellos apuntadores a los planos que se encuentran en el *convex hull* de la parte del sólido incluida en el nodo. De esta forma, los hijos necesitan representar únicamente a las caras fronterizas que no están en dicho convex hull y que forman las concavidades existentes.

El modelo descarta el tipo de nodo Negro ya que puede tratarlos exactamente como a un nodo Convexo en el que no hay planos de la frontera del sólido. Este modelo, a pesar de su capacidad de obtener representaciones más compactas que las del modelo Extendido, además de permitir la realización de operaciones Booleanas y la reconstrucción de las fronteras del sólido para permitir su visualización de formas más sencillas que en el modelo Extendido también, tiene la gran desventaja de ser un modelo aproximado, es decir, es incapaz de obtener representaciones exactas para poliedros en los que aparecen vértices compartidos por aristas tanto cóncavas como convexas, debiendo utilizar nodos Grises que se dividen recursivamente hasta la resolución mínima preestablecida.

2.7 Ventajas y desventajas de los modelos presentados

A continuación se resumen, de manera breve, las principales ventajas y desventajas de los modelos de representación geométrica descritos en el presente capítulo:

- El modelo de fronteras es uno de los más utilizados en la actualidad dada su gran facilidad de comprensión e implementación. Sin embargo, dado que por sí mismo es prácticamente incapaz de poder ser visualizado con eliminación de partes ocultas, resulta definitivamente impráctico para realizar operaciones Booleanas, y los algoritmos para manipularlo son casi siempre muy delicados en cuanto a la gran cantidad de casos extremos relacionados con errores de precisión matemática, casi siempre es acompañado de uno o más modelos geométricos auxiliares que le permiten superar sus desventajas. Y, ya que aún sin dichos modelos auxiliares es en realidad una de las representaciones menos concisas por la gran cantidad de información que se necesita para mantenerlo, la anexión de dichos modelos auxiliares lo convierte en un modelo extremadamente pesado en cuanto a cantidad de memoria necesaria, además del costoso tiempo de procesamiento que es requerido para convertirlo desde y hacia dichos modelos auxiliares.
- Los árboles BSP son un modelo muy poderoso y muy utilizado también. Su algoritmo de visualización soporta de manera natural la eliminación de partes ocultas, las operaciones Booleanas son mucho más simples que en el modelo de Fronteras, permite representaciones exactas, y son muy concisos dada la reducida cantidad de información que requieren. Sin embargo, para poliedros más complejos los árboles BSP resultantes

pueden llegar a ser de un gran tamaño debido a la gran cantidad de particiones que pueden sufrir sus caras, aún aplicando algoritmos de optimización [Paterson90]. Además, a diferencia de los OctTrees (tanto clásicos como Extendidos) donde los algoritmos de reconstrucción de fronteras y de operaciones Booleanas permiten operar a los nodos de manera independiente entre ellos (es decir, el resultado de operaciones entre dos octantes o de la reconstrucción de fronteras de uno de ellos no afecta bajo ninguna circunstancia las operaciones o reconstrucciones de los demás octantes, inclusive si se llegan a producir ligeros errores no perceptibles de precisión numérica), en los árboles BSP un nodo cuya frontera sea reconstruida con algún error de precisión numérica no perceptible o dos nodos procesados mediante alguna operación Booleana que presenten alguna falla descartable también de precisión numérica sí pueden y llegan a afectar de manera considerable las operaciones o reconstrucciones de los demás nodos. En otras palabras, sus algoritmos no son tan delicados a los errores de precisión matemática como en el modelo de fronteras, pero para poliedros complejos en realidad resulta muy común encontrarse con estos problemas. Por todo esto, es muy común encontrar este modelo siendo usado como auxiliar del modelo de fronteras, eliminando la necesidad de utilizar algoritmos de reconstrucción de las mismas, y utilizándolos únicamente como mecanismo de indexación del modelo de fronteras para permitir su visualización con eliminación de partes ocultas. Sin embargo, como ya se mencionó, esto produce representaciones con muy altos requerimientos de memoria.

- Los OctTrees clásicos tienen algunos de los algoritmos más sencillos y útiles para la realización de operaciones Booleanas y visualización con eliminación de partes ocultas, además de que su reconstrucción de fronteras para este último caso es muy sencilla también. Sin embargo, dado que son incapaces de obtener representaciones exactas

para ningún poliedro y que las estructuras obtenidas son generalmente muy grandes, resultan casi siempre imprácticos aún para poliedros simples, y nuevamente es más común encontrarlos únicamente como modelos auxiliares.

- Los PM-OctTrees, como ya se mencionó, resuelven las principales desventajas de los OctTrees clásicos, conservando a su vez la mayoría de sus ventajas. De esta manera, las estructuras obtenidas son mucho más pequeñas, y **casi** siempre es posible obtener representaciones exactas para cualquier poliedro. Sin embargo, como ya también se mencionó, en muchos casos sólo pueden obtener representaciones aproximadas, las operaciones Booleanas a nivel de los nodos Extendidos (especialmente cuando se consideran los nodos Vértice y Casi-Vértice) son casi tan complejas como en el modelo de Fronteras, la reconstrucción de las fronteras del modelo necesaria para su visualización presenta severas complicaciones a nivel de los nodos Vértice y Casi-Vértice, el soporte para nodos Vértice de *n* caras no es tan simple, y la visualización de nodos Vértice y Casi-Vértice requiere de mecanismos con alto costo de procesamiento para conseguir la eliminación de sus partes ocultas. Por todo esto, se considera a este modelo como una gran mejora al modelo clásico, pero siendo aún de cierta forma un paso intermedio hacia una mejor solución.

2.8 Propuesta del trabajo

Dada la necesidad de encontrar un modelo de representación geométrica que permita, de una manera eficiente, tanto la generación de su estructura, como la obtención de

representaciones exactas para cualquier poliedro *manifold* sin importar su complejidad, como su visualización con el consiguiente requerimiento de la reconstrucción de sus fronteras, y la realización de operaciones Booleanas, el presente trabajo propone una nueva y diferente extensión al modelo clásico de los OctTrees, denominada BSP-OctTree, que permite obtener dichas características, es decir, heredando las ya conocidas ventajas de dicho modelo, conservando al mismo tiempo las soluciones encontradas a sus principales desventajas en el modelo Extendido, y resolviendo los principales problemas de este último.

Además, el presente trabajo también propone que, para el caso específico de poliedros con una gran complejidad o un elevado número de caras, como son por ejemplo los datos geográficos del volcán Popocatépetl, que requieren ser visualizados o manipulados en diversas formas, este nuevo modelo tendrá la capacidad de procesarlos.

Así pues, en el siguiente capítulo, se describirá y detallará el modelo propuesto, los BSP-OctTrees, incluyendo tanto la descripción de su estructura, como el proceso requerido para su creación, y sus principales ventajas y desventajas.