Capítulo III
El modelo BSP-OctTree

3.1 Introducción

Como ya se mencionó **[Sección2.8]**, el presente trabajo propone una nueva y diferente extensión al modelo de OctTrees clásicos, la cual continúa heredando las ventajas de este modelo al mismo tiempo que resuelve de manera eficiente los principales problemas encontrados en el modelo Extendido preservando las ventajas que ya había adquirido. Esta extensión consiste en un nuevo tipo de nodo terminal, llamado nodo BSP. Este nuevo modelo, el BSP-OctTree, es básicamente un OctTree con cuatro tipos de nodos: nodo Blanco, nodo Negro, nodo BSP y nodo Gris. Los primeros dos representan un octante totalmente afuera o adentro respectivamente del sólido representado. Si el octante contiene un conjunto de caras cuyas relaciones topológicas sean simples, entonces el árbol BSP de todas las caras que intersectan al octante se almacena en un **nuevo tipo de nodo llamado nodo BSP**. De lo contrario, el octante procede a subdividirse recursivamente. El criterio inicial de "relación topológica simple" es si todas las caras que intersectan al octante comparten un vértice común, aún si dicho vértice no está en el octante analizado.

La razón principal del desarrollo del modelo propuesto es la de obtener una representación exacta y concisa que pueda realizar fácilmente operaciones Booleanas y pueda ser visualizada correctamente con eliminación de partes ocultas. Utilizar un nodo BSP como única extensión a los OctTrees clásicos permite obtener todas estas propiedades.

3.2 Creación de árboles BSP

Ya que, como ya se mencionó, la extensión propuesta utiliza árboles BSP, resulta importante detallar cómo puede crearse una estructura de este tipo de manera general. Básicamente, el algoritmo recibe como entrada una lista de caras describiendo de manera válida el modelo de fronteras de un poliedro cualquiera. En realidad, puede utilizarse cualquiera de las muchas variantes de modelo de fronteras que existen, siempre y cuando sea suficiente y forme una representación válida. En cada paso, el proceso recursivo selecciona una cara y particiona el resto del conjunto de caras C en tres subconjuntos: $C(H^+)$, $C(H^-)$ y C(H), que corresponden respectivamente a los tres subespacios H^+, H^- y H, definidos mediante la ecuación del plano de soporte de la cara elegida [Thibault87]. En otras palabras, dado el conjunto de caras C, éste se divide en tres subconjuntos, dependiendo de en qué lado de la ecuación de soporte de la cara elegida se encuentran todos los puntos internos de cada cara a clasificar, ya sea en el lado positivo, en el lado negativo, sobre el plano mismo, o bien formando parte de ambos lados, en cuyo caso dicha cara debe ser particionada por el plano y colocados ambos fragmentos resultantes en los subconjuntos apropiados. El conjunto C(H) es retenido y utilizado para crear un nuevo nodo para el árbol BSP, para posteriormente poder descartarse, y el proceso luego continúa recursivamente procesando la lista de caras del subconjunto $C(H^+)$ y $C(H^-)$, terminando cuando se procesan subconjuntos vacíos. Un punto importante es que, aunque no importa el mecanismo de selección utilizado para elegir una cara en cada paso recursivo, algunas

técnicas para la elección de la misma pueden producir árboles más deseables o concisos [Paterson90].

Así pues, el pseudocódigo para este procedimiento sería:

ALGORITMO 3.1
Creación de árboles BSP a partir de un modelo de fronteras.

```
procedure buildBSPTree(lista_de_caras) returns BSPTreeNode
 if (lista_de_caras está vacío) then
  return celda_hoja
 else
  elegir una cara C de lista_de_caras
  <lista_izquierda,lista_derecha,lista_coincidente> := particionar la
   caras en lista_de_caras utilizando H, agregando cada cara o fragmento
   a la lista apropiada
  nodo := nuevo nodo del árbol utilizando el plano de soporte H de C,
   considerando que todas las caras en lista_coincidente comparten H
  nodo.derecha := buildBSPTree(lista_derecha)
  nodo.izquierda := buildBSPTree(lista_izquierda)
  return nodo
 endif
endprocedure
```

3.3 Creación de BSP-OctTrees

3.3.1 Datos de entrada

Los requerimientos de entrada del modelo resultan ser prácticamente iguales a los de los OctTrees Extendidos [Argüelles00], [Ayala85], [Brunet85], [Navazo86]. Básicamente, utiliza como entrada una codificación en modelo de fronteras suministrando la siguiente información **[ApéndiceA]**:

- El número total de caras del objeto (cada una de las cuales puede estar delimitada por uno o más polígonos).
- El número de polígonos de cada cara (cada uno de los cuales estará delimitado a su vez por un conjunto de aristas que deberán conformar un contorno cerrado).
- El número de vértices de cada polígono de cada cara.
- La lista de las coordenadas (x, y, z) de los vértices de cada polígono de cada cara. Es importante mencionar que todos los polígonos asociados a una misma cara deberán ser coplanares, no deberán intersectarse entre sí, y el primero de ellos (que delimita el contorno externo de la cara) deberá contener dentro de sí mismo a los polígonos restantes (que delimitan los contornos de los posibles agujeros de la cara). Los vértices del contorno externo deberán ser valores reales ordenados en sentido contrario a las manecillas del reloj, mientras que los vértices de los contornos de los agujeros en el mismo sentido de las manecillas del reloj, todo esto visto desde el exterior del sólido. De esta manera, la normal de la cara apuntará hacia el exterior del objeto.

Además, también se requieren como datos de entrada:

- Las coordenadas (x, y, z) del origen del octante inicial (el universo).
- La longitud de arista del octante inicial (que deberá ser múltiplo de 2).
- La resolución mínima deseada para el OctTree, en caso de querer detener la recursión antes de lo requerido. Sin embargo, ya que el modelo sí permite obtener representaciones exactas para cualquier poliedro *manifold*, este valor puede omitirse.

3.3.2 Representación

La representación interna del esquema propuesto es similar a la usada en el modelo Extendido [Argüelles00], [Ayala85], [Brunet85]:

- Una tabla auxiliar es necesaria para almacenar las ecuaciones de los planos de soporte de las caras poligonales 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 $aX + bY + cZ + d < 0$.
- Ya que existen únicamente cuatro tipos diferentes de nodos (Blanco, Negro, Gris y nodo BSP), cada nodo necesita un identificador de 2 bits para diferenciarlos.
- Para los nodos BSP, su árbol BSP es almacenado internamente a continuación del identificador del nodo BSP. Cada nodo del árbol BSP tiene un bit que lo identifica ya sea como nodo hoja (0) o como nodo interno (1). Para nodos internos del árbol BSP, se almacena inmediatamente después de su identificador un apuntador a la ecuación del plano representada por dicho nodo en la tabla auxiliar de ecuaciones, para después almacenar la información de sus dos respectivos hijos.
- Si durante el proceso de construcción de un BSP-OctTree se llega a un octante que requiere descomposición recursiva, pero el valor deseado para la resolución mínima del árbol a obtener s se elige demasiado grande en lugar de ser omitido como se comentó en **[Sección3.3.1]**, un error de aproximación aparecerá. El modelo propuesto no soporta de manera directa la utilización de nodos Grises de Mínima Resolución, puesto que no los

necesita, pero este tipo de nodos pueden ser almacenados como nodos BSP con un árbol BSP nulo (usando el identificador del nodo BSP seguido inmediatamente por un bit 0).

Los nodos resultantes en el modelo propuesto no son siempre tan concisos como los nodos del modelo Extendido, y dichas diferencias pueden apreciarse en **[Tabla3.1]**. Aunque el identificador de los nodos ocupa menos espacio, aún así existe un incremento en ciertas situaciones topológicas debido a los bits que deben utilizarse como identificadores de los nodos de los árboles BSP que se encuentran representados dentro de los nodos BSP. Además, para aquellos nodos BSP donde su árbol BSP obtenido tuvo que particionar algunas de sus caras internas, la diferencia es más notoria. Sin embargo, es precisamente en estos últimos nodos donde el modelo Extendido es incapaz de obtener directamente visualizaciones correctas con eliminación de partes ocultas, y una posible solución para éste sería construir un árbol BSP cada vez que dichos nodos del modelo Extendido requieren ser visualizados, lo que el modelo propuesto evita totalmente, pues dichos árboles ya existen directamente en la representación. Además, para aquellos nodos BSP donde no se requirieron particiones de sus caras internas, es muy posible obtener nodos más concisos en la situación topológica de un vértice, comparando contra el modelo Extendido, ya que si se observa con cuidado **[Tabla3.1]** esto sucede siempre que $n<b$, lo que ocurre muy fácilmente en casi cualquier poliedro (excepto, por ejemplo, en los tetraedros o en la cúspide de las pirámides). Es decir, en la mayoría de los vértices de los objetos se observa la característica de que el número de caras incidentes en dicho vértice es casi siempre menor al logaritmo base 2 del número total de caras en el objeto. Por último, dado que el modelo propuesto puede detener la subdivisión recursiva en varios casos en los que el modelo Extendido no podía, el número total de nodos en los OctTree resultantes es

generalmente menor, consiguiendo en muchas ocasiones que a pesar del incremento individual en el tamaño de los nodos aún así la representación generada resulte más pequeña en comparación con el modelo Extendido **[Sección6.1]**.

TABLA 3.1
Comparación de los tamaños de nodos entre el Modelo Extendido y los BSP-OctTrees, para situaciones topológicas equivalentes.
b = bits requeridos para codificar un apuntador a la tabla de ecuaciones.
n = número de caras dentro del nodo (elaboración propia).

Situación topológica	Modelo Extendido	BSP-OctTree
Una cara	3 + b	5 + b
Una arista	4 + 2*b	7 + 2*b
Un vértice	3 + n* (b+1) + b	3 + n*(b+1) + n

Detallando los datos mostrados en **[Tabla3.1]**, en cuanto al modelo Extendido, los identificadores de sus nodos Extendidos requieren tres bits. Es por ello que los nodos Cara requieren los 3 bits del identificador, mas los *b* bits del apuntador de su cara (*3 + b*). Los nodos Arista requieren los 3 bits del identificador, mas los *b* bits del apuntador de cada una de sus dos caras, mas el bit de configuración (*3 + b + b + 1 = 4 + 2*b*). Los nodos Vértice requieren los 3 bits del identificador, mas *b* bits para almacenar el número de caras que convergen en el vértice, mas los *b* bits de cada uno de los apuntadores de sus *n* caras, más sus *n* bits de configuración, (*3 + b + n*b + n = 3 + n*(b+1) + b*).

En cuanto al modelo BSP-OctTree, los identificadores de sus nodos BSP requieren dos bits. Los nodos internos y hojas de un árbol BSP requieren 1 bit para su identificador. Es por ello que, en la situación topológica de una cara, se requieren los 2 bits del identificador, mas, respecto al árbol BSP asociado, el bit identificador de su nodo interno, mas los b bits del apuntador a la ecuación de soporte de dicho nodo interno, más los dos

bits necesarios para codificar los dos nodos hoja (*2 + 1 + b + 2 = 5 + b*). En cuanto a la situación topológica de una arista, se requieren los 2 bits del identificador, mas, respecto al árbol BSP asociado, los dos bits identificadores de sus nodos internos, mas los *b* bits de cada uno de los dos apuntadores a las ecuaciones de soporte de dichos nodos internos, mas los tres bits necesarios para codificar los tres nodos hojas (*2 + 2 + b + b + 3 = 7 + 2*b*). Por último, en cuanto a la situación topológica de un vértice, se requieren los 2 bits del identificador, mas, respecto al árbol BSP asociado (considerando que no se requirieron particiones de sus caras internas), los *n* bits identificadores de sus nodos internos, mas los b bits de cada uno de los n apuntadores a las ecuaciones de soporte de dichos nodos internos, mas los *n+1* bits necesarios para codificar los *n+1* nodos hoja (*2 + n + n*b + n + 1 = 3 + 2*n + n*b = 3+ n*(b+2) = 3 + n*(b+1) + n*).

3.3.3 Descripción del algoritmo

Nuevamente, el algoritmo de construcción del modelo resulta ser relativamente semejante al algoritmo utilizado para construir OctTrees Extendidos. Al igual que en el modelo Extendido, en el primer paso del algoritmo deben calcularse los coeficientes de las ecuaciones de soporte de las caras; luego, en un segundo paso, deben crearse dos listas: una con todas las caras que están completa o parcialmente dentro del octante inicial y otra con todos los vértices del poliedro que están también dentro del octante inicial [Argüelles00], [Ayala85], [Brunet85]. Por último, se llama al procedimiento que construye propiamente la estructura.

El algoritmo de construcción del árbol está basado en el siguiente procedimiento recursivo. Primero se considera el octante inicial y todas las caras que lo intersectan. Los octantes se subdividen recursivamente siempre que éstos no sean simples, y un octante se define como simple si éste es Blanco, Negro, o es un nodo BSP (también se considera como nodo BSP a aquellos nodos Grises alcanzan la resolución mínima especificada para el árbol, como se mencionó en **[Sección3.3.2]**). La forma de diferenciar estos nodos entre sí es básicamente por el número de caras que los intersectan.

De esta forma, cuando un octante está completamente afuera o adentro del objeto, es decir, no existe ninguna cara que lo intersecta, se clasifica como nodo Blanco o Negro respectivamente, al igual que en el modelo clásico. Cabe mencionar que en este caso no es posible determinar de primera instancia si el nodo es Blanco o Negro. El tipo de estos nodos puede ser determinado con precisión una vez que se ha encontrado el tipo de sus nodos hermanos, ya que ésta información puede extraerse de los mismos **[Sección3.3.6]**.

Ahora bien, cuando el conjunto de caras que intersectan un octante tienen un vértice común, se clasifica como nodo BSP, aún si el vértice mismo no se encuentra dentro de dicho octante. En este caso, un árbol BSP es construido y almacenado dentro del nodo usando el algoritmo descrito en **[Sección3.2]**, pero aplicando la pequeña optimización descrita en **[Sección3.3.5]**, utilizando el conjunto de caras que intersectan al octante. Es importante notar que el nodo BSP propuesto incluye todos los tipos de configuraciones diferentes que pueden aparecer en los nodos Cara, Arista, Vértice y Casi-Vértice del modelo Extendido, incluyéndolas a todas en un solo tipo de nodo. Además, aún los pocos casos especiales que conducirían a la creación de nodos Grises de Mínima Resolución en el

modelo Extendido también son incluidos, por lo que no existen errores de aproximación (por ejemplo, nodos Arista cuya arista común reside ligeramente afuera del octante, o nodos Casi-Vértice no incluyendo a todo el cono de caras incidentes en dicho vértice).

Como se mencionó, los nodos BSP están restringidos a ser creados únicamente cuando las caras que intersectan un octante determinado tienen un vértice común. La razón principal de esto consiste en que éste es el mínimo requerido para obtener representaciones exactas de cualquier poliedro, aún para aquellos muy complejos o con ángulos muy agudos. Por otro lado, si se relajara esta restricción, se obtendrían nodos BSP muy complejos que no recibirían ningún beneficio del esquema de OctTrees, además de adquirir los ya mencionados problemas en **[Sección2.7]** para árboles BSP complejos. Además, recordando lo mencionado en **[Sección1.2]** de que todo vértice en un objeto poliédrico *manifold* puede tener a lo más dos caras cóncavas incidentes en ese vértice en particular, si existe alguna cara de este tipo cuando un octante es clasificado como nodo BSP, puede ser elegida como la primera cara para construir el árbol BSP a fin de evitar particionarla por cualquier otra cara dentro del octante, reduciendo de ésta manera el tamaño final del árbol BSP obtenido. Esta estrategia reduce la probabilidad de obtener árboles BSP con caras particionadas, debido al hecho de que toda cara cóncava siempre es particionada si no es elegida primero, y si se relaja la restricción de crear nodos BSP únicamente cuando todas las caras tienen un vértice común, podría haber más de dos caras cóncavas dentro de un octante, y todas ellas serían particionadas con excepción de la primera. Por último, esta restricción permite aplicar la optimización propuesta en **[Sección3.3.5]** para el algoritmo general de creación de árboles BSP.

Así pues, el pseudocódigo para este procedimiento sería:

ALGORITMO 3.2
Construcción de BSP-OctTrees.

```
procedure buildOctTree(x,y,z,scale,minscale,type,lPol,lVrt,sign,oT)
 scaleq := scale/2
 for i:=1 to 8 do
  x1 := x+ax[i]*scaleq
  y1 := y+ay[i]*scaleq
  z1 := z+az[i]*scaleq
  determina subconjunto de caras y vértices dentro del octante i
  undetermined[i] := false
  if (num_caras=0) then
   addNode(B,oT)
   undetermined[i] := true
   sign[i] := -1
  else
   if (subconjunto de caras tienen un vértice común) then
    construye árbol BSP utilizando el subconjunto de caras
    addNode(BSP,oT)
    determina los signos de los dos vertices del octante i
   else
    if (scaleq>minscale) then
     addNode(G,oT)
     buildOctTree(x1,y1,z1,scaleq,minscale,type,lPol1,lVrt1,sgn1,oT)
     sign = sgn1
    else
     addNode(NULL,oT)
     determina los signos de los dos vértices del octante i
    endif
   endif
  endif
 endfor

 for i:=1 to 8 do
  if (undetermined[i]=true) then
   determina tipo del octante i de acuerdo a signos de hermanos
  endif
 endfor
endprocedure
```

donde los vectores *ax*, *ay*, *az* son:

ax = {0,1,0,1,0,1,0,1}
ay = {0,0,1,1,0,0,1,1}
az = {0,0,0,0,1,1,1,1}

de acuerdo al orden de recorrido de los nodos hijos sugerido por [Aguilera98], como variación del orden propuesto para los QuadTrees clásicos en [Samet90].

Para determinar el subconjunto de vértices que están dentro de un octante, simplemente se eligen todos los vértices cuyas coordenadas (x1, y1, z1) sean mayores o iguales a las coordenadas (*x*, *y*, *z*) del origen del octante, y estrictamente menores a (*x* + *scale*, *y* + *scale*, *z* + *scale*) respecto al origen del octante. El procedimiento **addNode** agrega un nodo al árbol codificado en código DF (B = blanco, G = gris, BSP = nodo BSP, NULL = nodo BSP con árbol vacío). Los demás detalles del pseudocódigo se explicarán en las siguientes secciones.

3.3.4 Determinación de las caras que intersectan un octante

Uno de los pasos más importantes para crear un BSP-OctTree consiste en determinar qué caras del conjunto recibido inicialmente intersectan al nodo hijo actual. Se considera que una cara intersecta a un nodo si éste contiene en su zona interior al menos un punto de la misma. Las fronteras izquierda, inferior y posterior se consideran como pertenecientes al nodo, mientras que las fronteras derecha, superior y frontal se consideran como no pertenecientes al nodo, y, de hecho, se consideran pertenecientes al correspondiente nodo vecino.

El procedimiento es idéntico al utilizado en el modelo Extendido, el cual es detallado con mucho más profundidad en [Argüelles00], [Navazo86]. Básicamente, se

deben aplicar las siguientes pruebas en forma consecutiva a cada uno de los elementos de un subconjunto de caras:

1) Detectar si al menos un vértice de la cara está dentro del nodo.
2) Usar una prueba *max-min* entre el nodo y el mínimo paralelepípedo que contiene a la cara.
3) Ver si el plano asociado a la cara intersecta al nodo.
4) Detectar si al menos una arista de la cara intersecta al nodo.
5) Ver si la intersección entre la cara y el nodo no es nula.

3.3.5 Creación de nodos BSP

Una vez que se ha determinado que en un subconjunto de caras todos sus elementos poseen un vértice común, debe procederse a crear un árbol BSP utilizando dicho subconjunto, como se mencionó en **[Sección3.3.3]**. Para ello, puede utilizarse directamente el algoritmo mostrado en **[Sección3.2]**. Sin embargo, a pesar de que los nodos BSP siempre definen regiones topológicas relativamente sencillas, es posible llegar a tener problemas de precisión numérica en casos extremos al intentar crear árboles BSP dentro de nodos BSP utilizando el algoritmo general. Esto se debe, principalmente, a que el área de los fragmentos de una o más caras que se encuentran dentro de un octante puede llegar a ser infinitamente pequeña, como es el caso de caras que tienen únicamente un punto en común con la frontera del octante pero que aún así se consideran dentro de él. En estos casos, el algoritmo general puede o bien desintegrar totalmente esos fragmentos al intentar particionarlos y por consiguientes no incluirlos en el árbol BSP resultante, o bien sí

considerarlos pero al mismo tiempo considerar cualquier fragmento infinitamente pequeño que pudiera producirse por errores de precisión numérica al intentar particionar cualquier cara.

Para resolver esto, se propone que el algoritmo general sí considere cualquier fragmento de cara, por pequeña que sea su área. Y para evitar que se tomen en cuenta fragmentos producidos por errores de precisión numérica, se propone la siguiente optimización al algoritmo general, la cual únicamente es posible gracias a la restricción establecida a los nodos BSP mediante la cual sólo son generados si todas las caras dentro del octante tienen un vértice en común. Dicha optimización consiste en que, al momento de determinar cuál es el vértice común de un conjunto de caras, al mismo tiempo dichas caras se ordenen de manera cíclica alrededor de dicho vértice, de manera semejante a como debe realizarse en el modelo Extendido. Desde luego, no es necesario que el cono completo de caras se encuentre presente dentro del octante, pues la única finalidad de la realización de este ordenamiento es para encontrar las caras que son adyacentes entre sí. En otras palabras, una cara cualquiera dentro de un nodo BSP tendrá a lo más 0,1 ó 2 caras vecinas. Utilizando dicha información en el procedimiento utilizado para particionar un subconjunto de caras por medio de la ecuación de un plano de soporte, puede evitarse intentar particionar cualquier cara que sea vecina a la cara cuyo plano de soporte se está utilizando como particionador, procediendo simplemente a clasificar un punto cualquiera de dicha cara con respecto a dicho plano para determinar de qué lado del plano se encuentra la cara. La justificación de esta optimización consiste en que dos caras vecinas dentro de un nodo BSP jamás se particionan entre sí. La única excepción a esto es cuando una de las dos caras es cóncava en el vértice común (puesto que es imposible que dos caras vecinas

sean cóncavas en su vértice común), pero si se elige siempre como primera de la lista cualquier cara cóncava que existiera, como se mencionó en **[Sección3.3.3]**, dicha cara cóncava queda sin particionar, y además dicha elección hace que cualquier otra cara cóncava quede particionada también en al menos dos caras convexas.

3.3.6 Determinación del signo de un punto respecto a un nodo

Como se mostró en **[Algoritmo3.2]**, es necesario en ciertas condiciones el poder determinar el signo de dos de los vértices de un octante. Esto significa que, dado un octante, deben elegirse dos de sus ocho vértices y determinar la posición de los mismos respecto a la parte del poliedro contenido dentro del nodo (es decir, si son internos o externos). La posición (o signo) de estos puntos se utiliza para determinar el tipo exacto de los nodos Blancos/Negros hermanos **[Sección3.3.3]**. Es decir, de los ocho vértices de un octante, uno de ellos coincide exactamente con la posición del punto medio de su nodo padre, mientras que otro de ellos coincide exactamente con uno de los ocho vértices de su nodo padre. El signo del vértice que coincide con el punto medio del nodo padre es el mismo signo de todos los nodos Blancos/Negros hermanos del nodo, por lo que al encontrar este signo puede determinarse con precisión el tipo de todos los nodos hermanos no definidos. El signo del vértice que coincide con uno de los ocho vértices del nodo padre se utiliza para devolverlo recursivamente de hijos a padres, ya que en un nivel superior este vértice puede coincidir con el punto medio de un nodo padre superior y utilizarse para definir los nodos hijos indefinidos del mismo [Argüelles00], [Ayala85].

Ahora bien, este procedimiento resultaba ser bastante complejo en el modelo Extendido. Para empezar, se requiere de un procedimiento especializado para cada tipo de nodo Extendido. El proceso ciertamente es casi directo cuando se consideran los nodos Cara y Arista. Pero para el caso de los nodos Vértice, Casi-Vértice o, peor aún, al llegar a nodos Grises de Mínima Resolución que no coincidieron con la configuración de ninguno de los nodos Extendidos soportados, la situación se torna muy difícil. Inclusive, esta fue una de las principales razones por las que originalmente sólo fueron soportados nodos Vértice de 3 ó 4 caras incidentes: se utilizaba una tabla que enumeraba todas las posibles configuraciones de nodos con 3 ó 4 caras, y la respectiva operación a aplicar para calcular el signo de un punto con respecto a dichas configuraciones. Y, aunque posteriormente se resolvió dicha limitación, el proceso continuó siendo complejo.

El problema principal reside nuevamente en una limitación inherente al modelo de fronteras. Aunque es trivial calcular el signo de un punto con respecto a la ecuación de soporte de una cara cualquiera, lo que no resulta tan trivial es determinar en qué forma deben combinarse los resultados de dichas sustituciones con respecto a un conjunto de caras para determinar el signo de un punto respecto a un sólido. Para el caso 2D, existe una fórmula muy simple mostrada en [Dobkin88] que utiliza únicamente la operación XOR binaria y el complemento para combinar dichos resultados, ordenando los resultados de las sustituciones del punto en las ecuaciones de soporte de las aristas del polígono en sentido contrario a las manecillas del reloj, siendo que la fórmula a utilizar depende únicamente de si el ángulo formado por las aristas adyacentes es cóncavo o convexo. Pero no existe nada ni remotamente semejante para el caso 3D. Si el sólido es convexo, los signos de las sustituciones deben combinarse simplemente utilizando la operación AND binaria. De no

ser así, normalmente es más sencillo convertir el modelo a alguna otra representación que permita obtener la fórmula para combinar los resultados de las sustituciones, como por ejemplo puede ser una representación CSG que utilice únicamente semiespacios [Damski88], [Hartquest94], [Shapiro93], [Shapiro97]. O bien, puede procederse a intentar descomponer el sólido en sus componentes convexas [Edelsbrunner95], [Schewchuk99], para posteriormente combinar con AND binario los resultados de las sustituciones en cada componente convexo, y al final combinar con OR binario los resultados ya combinados. Una última opción puede consistir en operar directamente con el modelo de fronteras, pero en lugar de intentar combinar sustituciones de puntos en ecuaciones de soporte, utilizar métodos como los descritos en [Franklin1], [Franklin2], [Said].

Sin embargo, ya que el modelo propuesto utiliza a los nodos BSP como única extensión al modelo clásico, este procedimiento no sólo se simplifica desde el punto de vista de tener que utilizar únicamente un caso, sino que, además, calcular el signo de un punto respecto a un nodo BSP se reduce a calcular el signo de dicho punto con respecto al árbol BSP contenido en dicho nodo, lo cual puede resolverse sencillamente con el siguiente algoritmo [Thibault87]:

ALGORITMO 3.3
Clasificación de un punto respecto a un árbol BSP.

```
procedure pointClassify(punto,BSPNode) returns {in,out,on}
 if (BSPNode es nodo hoja) then
  return valor del nodo hoja (in o out)
 else
  d := productoPunto(punto,BSPNode)
  if (d<0) then
   return pointClassify(punto,BSPNode.izquierda)
  else if (d>0) then
   return pointClassify(punto,BSPNode.derecha)
```

```
   else
    l := pointClassify(p,BSPNode.izquierda)
    r := pointClassify(p,BSPNode.derecha)
    if (l=r) then
     return r
    else
     return on
    endif
   endif
  endif
endprocedure
```

donde **productoPunto** es un procedimiento que obtiene el signo del punto respecto a la ecuación del plano de soporte contenido en el nodo actual del árbol BSP.

Por último, es importante mencionar que, cuando el signo del centro de un nodo padre indica que dicho punto se encuentra exactamente sobre la superficie del poliedro contenido en dicho nodo, en el modelo Extendido se presentaba nuevamente un gran problema que debía ser manejado por medio de tablas especiales o algún otro mecanismo alternativo. Esto se debe, principalmente, a que en estos casos en particular no todos los nodos Blancos/Negros hermanos serán forzosamente del mismo tipo, por lo que el signo del centro del nodo padre no es suficiente. Sin embargo, cuando esto sucede en el modelo propuesto, puede resolverse calculando el color de cada uno de los nodos Blancos/Negros hermanos por separado, simplemente encontrando el signo del punto del centro de cada uno de estos nodos hermanos utilizando el árbol BSP del poliedro completo, que puede ser creado una sola vez antes de iniciar la creación del BSP-OctTree y desechado posteriormente.

3.4 Ventajas y desventajas

Usando el modelo descrito en el presente capítulo es posible resolver los problemas encontrados en el modelo Extendido:

- Se puede obtener una representación exacta para cualquier objeto poliédrico *manifold*.
- Sólo se requiere un algoritmo para realizar operaciones Booleanas **[CapítuloV]** sin casos especiales y todavía recorriendo ambos árboles al mismo tiempo.
- Aún para vértices complejos con caras incidentes cóncavas en dicho vértice puede ser posible la reconstrucción de sus fronteras con el fin de ser visualizados **[Sección4.2.1]**.
- Clasificar un punto P con respecto a un nodo BSP es muy sencillo **[Sección3.3.6]**.
- Un ordenamiento espacial es inducido dentro de los nodos BSP, así que visualizarlos con eliminación de partes ocultas ya no es un problema **[CapítuloIV]**.

La desventaja más importante del modelo radica en la debilidad de los árboles BSP ante los errores de precisión numérica. Aunque gran parte de esta debilidad es resuelta mediante la optimización propuesta en **[Sección3.3.5]**, aún quedan algunos pocos casos en los que pueden obtenerse errores de precisión al intentar obtener la representación del árbol BSP en un nodo BSP, errores que son visualmente perceptibles al intentar reconstruir las fronteras de dichos árboles para visualizarlos. Este tipo de errores no se presentaban en el modelo Extendido, ya que visualmente cualquier error de precisión numérica al intentar reconstruir las fronteras de un nodo Extendido no era perceptible.