Capítulo 3: Capítulo 3: El Modelo BSP-OctTree

3.1.Introducción

En [Argüelles02] se presentó un nuevo modelo, el cual es una extensión del modelo OctTree clásico, el cual continúa heredando las ventajas de este modelo, y además resuelve los problemas del modelo extendido preservando las ventajas ya adquiridas con él. Esta extensión consiste en un nuevo tipo de nodo Terminal, llamado nodo BSP. Este nuevo modelo, llamado BSP-OctTree, es básicamente un OctTree con cuatro tipos de nodos: nodo Blanco, Negro, Gris y BSP. Los dos primeros representan un octante completamente fuera o dentro del objeto, respectivamente. 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 cortante procede a subdividirse recursivamente. Cuando hablamos de una "relación topológica simple" nos referimos a que todas las caras que intersectan al octante comparten un vértice en común, aún si dicho vértice no está dentro del octante en cuestión [Argüelles02].

Este nuevo modelo nos permite obtener una representación exacta y concisa que puede realizar operaciones Booleanas y puede ser visualizada correctamente con eliminación de partes ocultas.

3.2.Creación de BSP-OctTrees

3.2.1. Datos de entrada

La entrada para este modelo resulta ser prácticamente igual al de los OctTrees Extendidos [Argüelles00], [Ayala85], [Brunet85], [Navazo86]. Básicamente utiliza como entrada una codificación en modelo de fronteras suministrando la siguiente información.

- 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. Cabe 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.
- 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.

3.2.2. Representación

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

- Una tabla auxiliar, 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 $az + by + cz + d < 0$.
- Siendo que únicamente existen cuatro tipos diferentes de nodos (Blanco, Negro, Gris y BSP) cada nodo necesita un identificador de 2 bits para diferenciarlos.
- En el caso de 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 se elige demasiado grande en lugar de ser omitido, aparecerá un error de aproximación. 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.

3.3.Algoritmo de construcción

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; 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. [Argüelles02].

El algoritmo de construcción del árbol está basado en el siguiente procedimiento recursivo. Primero consideramos 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. La forma de

diferenciar estos nodos entre sí es básicamente por el número de caras que los intersectan [Argüelles02].

Cuando un octante está completamente fuera o dentro del objeto, es decir, cuando ninguna cara interfecta al octante, se clasifica como nodo Blanco o Negro respectivamente. Aunque en este caso no es posible determinar de primera instancia si el nodo es Blanco o Negro. El tipo de estos nodos será determinado con precisión cuando se haya encontrado el tipo de nodo de sus hermanos. [Argüelles02]

En el caso de que las caras que intersectan el octante tengan un vértice en 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 las caras que intersectan al nodo. Es importante notar que el nodo BSP engloba todos los tipos de configuraciones del modelo extendido (Cara, Arista, Vértice y Casi-Vértice). Aún los casos especiales que conducirían a la inclusión de un nodo Gris de Mínima Resolución, también son incluidos, por lo que no existen errores de aproximación. [Argüelles00].

Así pues, el pseudo código del procedimiento seria:

Algoritmo 3.1: Construcción de BSP-OctTrees [Argüelles02].

```
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].

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, N=Negro, G=Gris, BSP=nodo BSP, NULL=nodo BSP vacío). [Argüelles02]

Para determinar si una cara dada intersecta al octante, debemos verificar si al menos un punto de la cara está en la zona interior del octante. Las fronteras izquierda, inferior y posterior se consideran como parte del octante, mientras que las fronteras derecha, superior y frontal se consideran no pertenecientes al nodo. El procedimiento para lograr esto es idéntico al usado en el modelo Extendido [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 *min-max* 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.

3.3.1. Adición de un nodo BSP

Ya que se determino que un subconjunto de caras del objeto intersectan un mismo octante y que además tienen todas un vértice en común, debe procederse a la inserción de un nodo BSP, lo que implica la construcción de un árbol BSP. Para tal motivo resulta necesario mostrar de manera general el algoritmo de creación de un árbol BSP.

Básicamente, el algoritmo recibe una lista de caras describiendo de manera válida el modelo de fronteras de un poliedro. 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 a los tres subespacios creados H^+, H^- y H, definidos mediante la ecuación del plano de soporte de la cara elegida [Thibault87]. En caso de que al aplicar el particionador, una cara resida en ambos lados de él, se procede a particionar dicha cara por el plano y colocando los fragmentos resultantes en el lado correspondiente. El conjunto C(H) es retenido y usado APRA crear un nuevo nodo para el árbol BSP, y el proceso luego continúa recursivamente procesando la lista de caras del subconjunto $C(H^+)$ y $C(H^-)$ hasta agotar toda la lista. Es importante señalar que el mecanismo de

selección del particionador, puede generar árboles más deseables o concisos [Paterson90].

Algoritmo 3.2: Creación de árboles BSP a partir de un modelo de fronteras [Argüelles02].

```
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
```

A pesar de que los nodos BSP siempre definen regiones topológicas relativamente sencillas, es posible llegar a tener problemas de precisión en casos extremos, al intentar crear árboles BSP; debido 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 las caras que tienen únicamente un punto en común con la frontera del octante. En estos casos el algoritmo general puede llegar a desintegrar tales fragmentos. Por tal motivo en [Argüelles02] se propone la siguiente mejora al algoritmo general, la cual únicamente es posible gracias a la restricción establecida a los nodos BSP donde éstos son solo generados si las caras tienen un vértice en común. La optimización consiste en que al momento de determinar cuál es el vértice común de un conjunto de caras, al mismo tiempo dichas caras se ordenan de manera cíclica alrededor de dicho vértice, de manera semejante a como se hace en el modelo Extendido. Esto con la finalidad de realizar un ordenamiento para encontrar las caras que son adyacentes entre si. Con este ordenamiento 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 que lado del plano se encuentra la cara [Argüelles02].

3.3.2. Determinación del signo de un punto respecto a un nodo

Una vez que se concluyó con la creación del árbol, es necesario, en ciertas ocasiones, 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. La posición de estos puntos son usados para determinar el tipo exacto de los nodos Blanco/Negros hermanos. 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 usarse para definir los nodos hijos indefinidos del mismo [Argüelles02], [Argüelles00], [Ayala85].

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 se resuelve 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.

3.4.Visualización

El modelo BSP-OctTree hereda tanto la propiedad de los OctTrees clásicos de poder ser visualizados con eliminación correcta de partes ocultas, gracias a su estructura regular compuesta de cubos que no se intersectan entre sí; como las cualidades de los árboles BSP. Lo cual nos deja únicamente con el problema de calcular las coordenadas correctas en 2D, es decir, la traslación y la proyección de un objeto en pantalla, y respecto a cualquier efecto visual; para lo cual existen infinidad de mecanismos descritos en la literatura. [Argüelles02]

Para poder visualizar un árbol BSP debe determinarse la posición relativa del observador con respecto al nodo. Si el observador está frente al nodo, el hijo izquierdo

debe ser visualizado recursivamente primero, luego el nodo en sí, y por último el hijo derecho. Siendo el recorrido inverso si el observador se encuentra atrás del nodo.

Finalmente, el algoritmo requerido para visualizar el BSP-OctTree sigue la forma descrita para los OctTree Clásicos, dependiendo de la posición del observador con respecto al universo. Cuando se llega a un nodo BSP, puede ser visualizado siguiendo el recorrido de su árbol usando la misma posición del observador. Los nodos Blancos no necesitan ser visualizados ya que son eternos al objeto, y los nodos Negros tampoco necesitan ser visualizados ya que éstos son internos al objeto y quedan cubiertos por los nodos BSP.

La posición relativa del observador puede calcularse una única vez antes del proceso de visualización, usando los ángulos de rotación respecto a los ejes x, y, z que se deseen, mediante la multiplicación de las coordenadas de la posición inicial del observador por la siguiente matriz [Argüelles00], [Argüelles02].

$$\left[\begin{array}{c|c|c} \cos(y)\cos(z) & \cos(x)\,\mathrm{sen}(z)+\mathrm{sen}(x)\,\mathrm{sen}(y)\cos(z) & \mathrm{sen}(x)\,\mathrm{sen}(z)-\cos(x)\,\mathrm{sen}(y)\cos(z) \\ -\cos(y)\,\mathrm{sen}(z) & \cos(x)\cos(z)-\mathrm{sen}(x)\,\mathrm{sen}(y)\,\mathrm{sen}(z) & \mathrm{sen}(x)\cos(z)+\cos(x)\,\mathrm{sen}(y)\,\mathrm{sen}(z) \\ \mathrm{sen}(y) & -\mathrm{sen}(x)\cos(y) & \cos(x)\cos(y) \end{array}\right]$$

3.5.Operaciones Booleanas

Una vez más, las operaciones Booleanas entre BSP-OctTrees están basadas directamente en los usados en el modelo de OctTrees clásicos. Los únicos cambios requeridos aparecen cuando uno de los nodos a operar es un nodo BSP, en cuyo caso se usan los algoritmos descritos a continuación.

3.5.1. Operaciones Booleanas entre árboles BSP

Las operaciones Booleanas binarias (intersección, unión y diferencia) pueden ser construidas sobre un procedimiento más general conocido como fusión de árboles BSP. Esta operación es completamente independiente del tipo de operación Booleana concreta a realizar, ya que esta distinción no es necesaria durante todo el proceso excepto cuando se alcanzan nodos hoja, en cuyo caso debe utilizarse un pequeño procedimiento que determine el paso a realizar dependiendo de la operación elegida. [Argüelles02]

3.5.1.1. Complemento

Para obtener el complemento de un árbol BSP se debe realizar un recorrido a través de sus nodos, siguiendo los siguientes criterios [Argüelles02]:

- Si el nodo alcanzado es una celda, simplemente se complementa su valor. Es decir, las celdas in se convierten en out y viceversa.
- Si el nodo alcanzado es interno, simplemente se invierten los dos apuntadores a sus hijos. Es decir, su hijo izquierdo se convierte en el derecho y viceversa.
- Se modifican las orientaciones de las ecuaciones de los planos de soporte. Esto se realiza cambiando los signos de los coeficientes de cada una de las ecuaciones en la tabla auxiliar asociada a la representación.

3.5.1.2.Fusión de árboles BSP

Fusionar dos árboles BSP significa que, dadas dos particiones del mismo espacio, P1 y P2, se forma una nueva partición **P3** = **P1** + **P2** mediante la combinación de las celdas de P1 y P2, por ejemplo, una celda **c3** $\in$ **P3** $\Leftrightarrow$ $\exists$ **c1** $\in$ **P1**, **c2** $\in$ **P2**, dado que **c3** = **c1** $\cap$ **c2**, **c3** $\neq$ Ø [Naylor90]. Fusionar puede ser ilustrado mediante la simple sobreposición de dos particiones, una encima de la otra [Figura3.1].

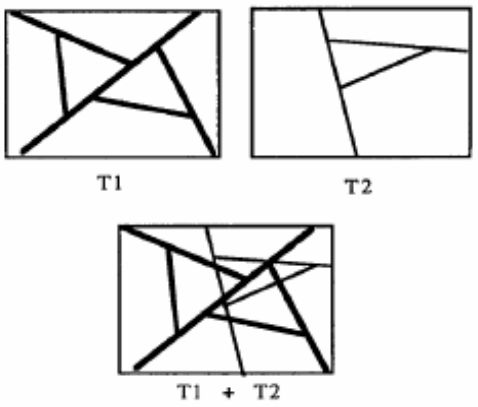

Figura 3.1: Fusionando particiones [Naylor90] (Traducción).

La fusión puede entenderse en términos del paradigma de insertar un objeto en un árbol; en este caso el objeto es un árbol también. Para ello, se necesitan dos operaciones básicas: realizar una partición binaria del objeto si se llega a un nodo interno, y ejecutar una operación celda-objeto si se llega a un nodo hoja. Realizar la partición binaria de un árbol BSP por medio del particionador binario de un nodo interno produce dos nuevos árboles. La operación celda-árbol es la que determina la operación Booleana específica a realizar. Su función es la de combinar los atributos de una celda con los atributos de un árbol, y el resultado será ya sea la celda o el árbol mismos. En la [figura 3.2] puede verse la fusión de dos árboles. [Argüelles02]

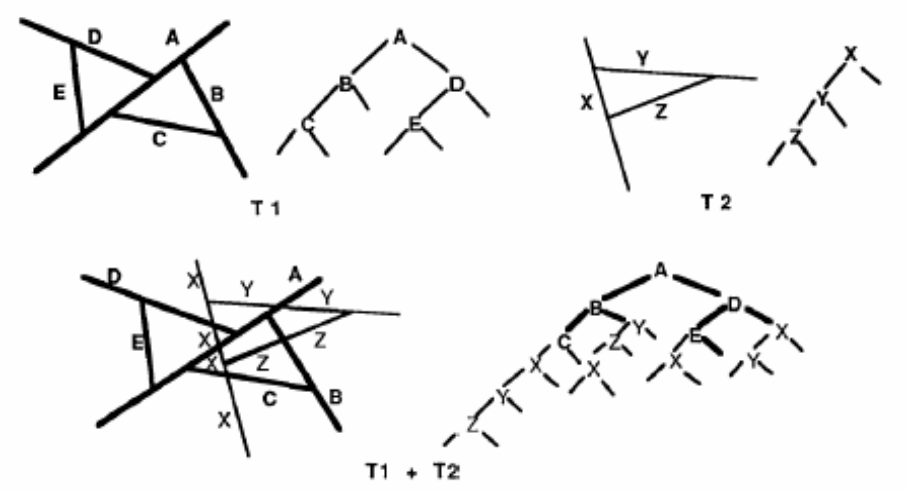

Figura 3.2: Fusionando dos árboles BSP [Naylor90].

El pseudocódigo de este procedimiento sería [Naylor90]:

Algoritmo 3.4: Fusión de árboles BSP.

```
procedure mergeBSPTrees(T1,T2) returns BSPTree
 if (T1 es nodo hoja) or (T2 es nodo hoja) then
  nodo := mergeTreeWithCell(T1,T2)
 else
  T2partitioned := partitionBSPTree(T2,T1.H)
  nodo.H- := mergeBSPTrees(T1.H-,T2partitioned.H-)
  nodo.H+ := mergeBSPTrees(T1.H+,T2partitiones.H+)
  nodo.H := T1.H
 endif
 return nodo
endprocedure
```

donde **mergeTreeWithCell** es el procedimiento celda-árbol y **partitionBSPTree** es la partición binaria de un árbol BSP.

3.5.1.3.Intersección unión y diferencia

Ya que está definido el proceso de fusión de árboles BSP, realizar operaciones Booleanas con ellos se vuelve una tarea relativamente fácil. El proceso de fusión continúa recursivamente hasta que uno de los dos operandos es una celda. Cuando

alcanzamos esta condición, debemos fusionar los atributos de la celda con aquellos de otro árbol arbitrario. Esto implica seleccionar simplemente ya sea la celda o el árbol, posiblemente complementado. El pseudocódigo para esta tarea es el siguiente [Argüelles02]:

Algoritmo 3.5: Procedimiento celda-árbol para operaciones Booleanas.

```
procedure mergeTreeWithCell (T1,T2) returns BSPTree
 if (T1 es celda in) then
  case operacion of
   union: return T1
   interseccion: return T2
   diferencia: return complement(T2)
  endcase
 elseif (T1 es celda out) then
  case operacion of
   union: return T2
   interseccion: return T1
   diferencia: return T1
  endcase
 else
  repetir el bloque anterior pero con T1 y T2 invertidos
 endif
endprocedure
```

donde **complement** obtiene el complemento del un árbol BSP.

3.5.2. Operaciones Booleanas entre BSP-OctTrees

3.5.2.1.Complemento

Para complementar un árbol BSP-OctTree es preciso realizar un recorrido por el árbol según los siguientes lineamientos [Navazo86], [Argüelles00], [Argüelles02]:

- Los nodos Blanco se transforman en Negros y viceversa
- Los nodos Grises permanecen inalterados

- Los nodos BSP se complementan usando el algoritmo ya descrito anteriormente. Si el nodo BSP representará un árbol nulo éste permanece inalterado.

En el primer paso del algoritmo se copia la tabla auxiliar de las ecuaciones de soporte, pero cambiando los signos de todos los coeficientes de las mismas. Después se llama al procedimiento que construirá el nuevo árbol a partir de la codificación DF del árbol original.

Algoritmo 3.6: Complemento de BSP-OctTrees [Argüelles02].

```
procedure complementBSPOctTree(x,y,z,scale,oT,newOT)
 scaleq := scale/2
 for i:=1 to 8 do
  x1 := x+ax[i]*scaleq
  y1 := y+ay[i]*scaleq
  z1 := z+az[i]*scaleq
  nodo := getType(oT)
  case nodo of
   Negro:
    addNode(B,newOT)
   Blanco:
    addNode(N,newOT)
   Gris:
    addNode(G,newOT)
    complementBSPOctTree(x1,y1,z1,scaleq,oT,newOT)
   BSP:
    arbol := complementBSP(nodo.bsp)
    addNode(arbol,newOT)
  endcase
 endfor
endprocedure
```

donde *ax*, *ay*, *az* son los mismos que en la construcción del BSP-OctTree.

3.5.2.2.Intersección

El procedimiento de intersección de dos BSP-OctTrees, igualmente, es una variable del modelo clásico. Se examinan los nodos de cada árbol de manera sincronizada y

simultánea. Así, en el caso de que el siguiente nodo de cada árbol fuese un nodo hoja, las hojas son intersectadas. Si uno de los árboles tiene un nodo Gris, y el otro tiene un nodo BSP, entonces el recorrido del primer árbol continúa mientras el otro se queda detenido, y las intersecciones en estos casos son manejadas entre nodos de diferente tamaño. Los criterios de intersección de los nodos clásicos siguen siendo los mismos del modelo clásico, sin embargo, cuando tenemos que intersectar un nodo BSP se toman los siguientes criterios [Argüelles02]:

- La intersección entre dos nodos BSP requiere únicamente de operar los árboles BSP asociados a ambos nodos con los algoritmos de intersección de los árboles BSP. El resultado de la operación será el nuevo nodo BSP; en caso de que el poliedro descrito por el árbol no cumpla con la restricción de que todas sus caras tengan un vértice en común, se procederá a reconstruir el poliedro y dividirlo recursivamente para crear nodos validos.
- La intersección entre un nodo Gris y un nodo BSP no es inmediata, dado que se desconoce de forma directa el interior del nodo Gris. Como salida se generará un nodo Gris y se efectuará directamente la intersección de los descendientes del nodo Gris con el nodo BSP. Este método implica la intersección entre nodos de diferentes tamaños.

El primer paso del algoritmo consiste en “pegar” las dos tablas de ecuaciones de soporte de los árboles, la segunda al final de la primera, por lo que hay que tener cuidado al momento de operar los árboles de reasignar los apuntadores de los nodos del segundo árbol. Después se llama al procedimiento principal que construye el nuevo árbol a partir de la codificación DF de los dos árboles originales:

Algoritmo 3.7: Intersección de BSP-OctTrees [Argüelles02].

```
procedure intersectBSPOctTrees(x,y,z,scale,oT1,oT2,newOT)
 scaleq := scale/2
 for i:=1 to 8 do
  x1 := x+ax[i]*scaleq
  y1 := y+ay[i]*scaleq
  z1 := z+az[i]*scaleq
  nodo1 := getType(oT1)
  nodo2 := getType(oT2)
  if ((nodo1=Blanco) or (nodo2=Blanco)) then
   addNode(B,newOT)
   saltarse los descendientes si uno de los dos nodos es Gris
  else
   if ((nodo1.bsp=NULL) or (nodo2.bsp=NULL)) then
    addNode(NULL,newOT)
    saltarse los descendientes si uno de los dos nodos es Gris
   else
    if (nodo1=Negro) then
     addNode(nodo2,newOT)
     copiar la descendencia al resultado si el otro nodo es Gris
    else
     if (nodo2=Negro) then
      addNode(nodo1,newOT)
      copiar la descendencia al resultado si el otro nodo es Gris
     else
      if (nodo1=BSP) and (nodo2=BSP) then
       arbol := mergeBSPTrees(nodo1.bsp,nodo2.bsp)
       if (arbol cumple restriccion) then
        addNode(arbol,newOT)
       else
        recuperar fronteras de arbol y generar lPol,lVrt
        buildOctTree(x1,y1,z1,scaleq,minscale,type,lPol,lVrt,sign,O)
        addNode(G,newOT)
        addNode(O,newOT)
       endif
      else
       addNode(G,newOT)
       if (nodo2=BSP) then
        intersectGreyBSP(x1,y1,z1,scaleq,oT1,nodo2.bsp,newOT)
       else
        if (nodo1=BSP) then
         intersectGreyBSP(x1,y1,z1,scaleq,oT2,nodo1.bsp,newOT)
        else
         intersectBSPOctTrees(x1,y1,z1,scaleq,oT1,oT2,newOT)
        endif
       endif
      endif
     endif
    endif
   endif
  endif
 endfor
endprocedure
```

donde *ax, ay , az* son los mismos usados anteriormente. Los procedimientos **mergeBSPTrees** y **buildOctTree** ya fueron descritos con anterioridad.

El procedimiento **intersectGrayBSP** es el que se encarga de la intersección de un nodo Gris con un BSP, que son de diferente tamaño. Este procedimiento sigue los siguientes criterios [Argüelles02]:

- Cuando se operan nodos BSP de tamaños diferentes, el nodo BSP de mayor tamaño debe ser intersectado primero con otro árbol BSP que describa un cubo del tamaño del nodo BSP menor, para después operar el resultado directamente con el nodo BSP menor. El resultad de esta segunda operación se debe agregar al resultado como un nodo BSP si cumple que todas sus caras tienen un vértice en común, o subdividirse recursivamente.
- Cuando se opera un nodo BSP con un nodo negro de menor tamaño, el nodo BSP también debe ser intersectado con un árbol BSP que describa un cubo del tamaño del nodo negro, agregando el resultado de dicha intersección como un nodo BSP en el árbol de salida.

El pseudocódigo de este procedimiento sería [Argüelles02]:

Algoritmo 3.8: Intersección de un nodo BSP con un árbol BSP-OctTree.

```
procedure intersectGreyBSP(x,y,z,scale,oT,bspRoot,newOT)
 scaleq := scale/2
 for i:=1 to 8 do
  x1 := x+ax[i]*scaleq
  y1 := y+ay[i]*scaleq
  z1 := z+az[i]*scaleq
  nodo := getType(oT)
  if (nodo=Blanco) then
    addNode(B,newOT)
  else
   if (nodo.bsp=NULL) then
    addNode(NULL,newOT)
   else
    if (nodo=Gris) then
     addNode(G,newOT)
```

```
      intersectGreyBSP (x1,y1,z1,scaleq,oT,bspRoot,newOT)
     else
      if (nodo=Negro) then
       arbol := mergeBSPTrees(bspRoot,cubo de tamaño scaleq)
       addNode(arbol,newOT)
      else
       temp := mergeBSPTrees(bspRoot,cubo de tamaño scaleq)
       arbol := mergeBSPTrees(temp,nodo.bsp)
       if (arbol cumple restriccion) then
        addNode(arbol,newOT)
       else
        recuperar fronteras de arbol y generar lPol,lVrt
        buildOctTree(x1,y1,z1,scaleq,minscale,type,lPol,lVrt,sign,O)
        addNode(G,newOT)
        addNode(O,newOT)
       endif
      endif
     endif
    endif
   endif
  endfor
endprocedure
```

donde *ax*, *ay*, *az* son los ya antes mencionados.

3.5.2.3.Unión y diferencia

Las operaciones de unión y diferencia en el modelo BSP-OctTree pueden definirse modificando la tabla de intersección del modelo clásico según sea necesario, sin necesidad de aplicar cambios a la forma como se manipulan los nodos BSP, o bien combinando las operaciones de intersección y complemento, de la forma:

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

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

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