Capítulo V
Operaciones Booleanas

5.1 Introducción

Es muy posible que en muchos casos sea necesario comparar dos objetos y determinar cuál es su parte común. Esto implica intersectar los dos objetos y determinar qué puntos son comunes a ambos. Otras operaciones que son útiles son el complemento, la unión y diferencia.

En este capítulo de presentan los algoritmos para la realización de las operaciones Booleanas de complemento e intersección entre poliedros codificados en el modelo de BSP-OctTrees, ya que éstas dos operaciones permiten el cómputo directo del resto (unión y diferencia), como se mencionó en **[Sección2.4.4]**. Para ello, también será necesario presentar los algoritmos homólogos utilizados en el modelo BSP, ya que éstos son requeridos para operar los nodos BSP.

5.2 Operaciones Booleanas entre árboles BSP

Los algoritmos necesarios para la realización de operaciones Booleanas entre árboles BSP son relativamente simples y recursivos. En especial, para las operaciones Booleanas binarias (intersección, unión y diferencia), éstas 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 Booleana elegida.

5.2.1 Complemento

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

- 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, el hijo izquierdo se coloca en el apuntador 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.

5.2.2 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 ≠ Ø [Naylor90]. Fusionar puede ser ilustrado mediante la simple sobreposición de dos particiones, una encima de la otra **[Figura5.1]**.

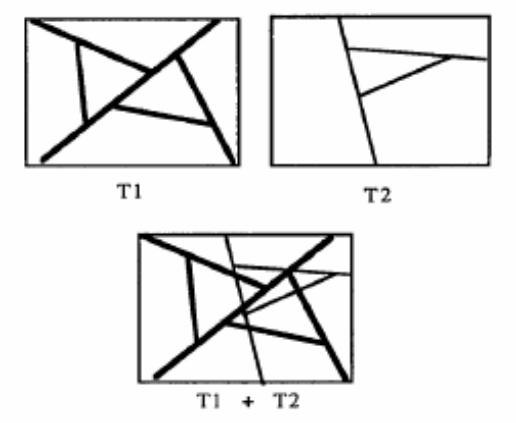

FIGURA 5.1
Fusionando particiones [Naylor90] (Traducción).

Como en la mayoría de los algoritmos que manipulan árboles BSP, la fusión puede entenderse en términos del paradigma de insertar un objeto en un árbol; en este caso, el objeto es un árbol también. 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 **[Sección5.2.3]**. La operación celda-árbol es precisamente 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 **[Sección5.2.4]**.

Dadas estas dos operaciones, el algoritmo particiona un árbol T2 por medio del particionador binario de la raíz de otro árbol T1. Los dos árboles resultantes, $T2^-$ y $T2^+$,

están definidos exactamente en la misma región que $T1.H^-$ y $T1.H^+$. Luego entonces, han sido creados dos nuevos subproblemas, cada uno de los cuales idéntico en forma al problema original: fusionar dos árboles, cada uno de los cuales particiona el mismo subespacio. Cuando una celda es alcanzada, la operación celda-árbol es llamada, la que como ya se mencionó depende de la operación Booleana a realizar. En **[Figura5.2]** se puede observar la fusión de dos árboles. Como se puede observar, cada nodo celda de T1 es reemplazado por el subconjunto de T2 que reside en la región de dicha celda.

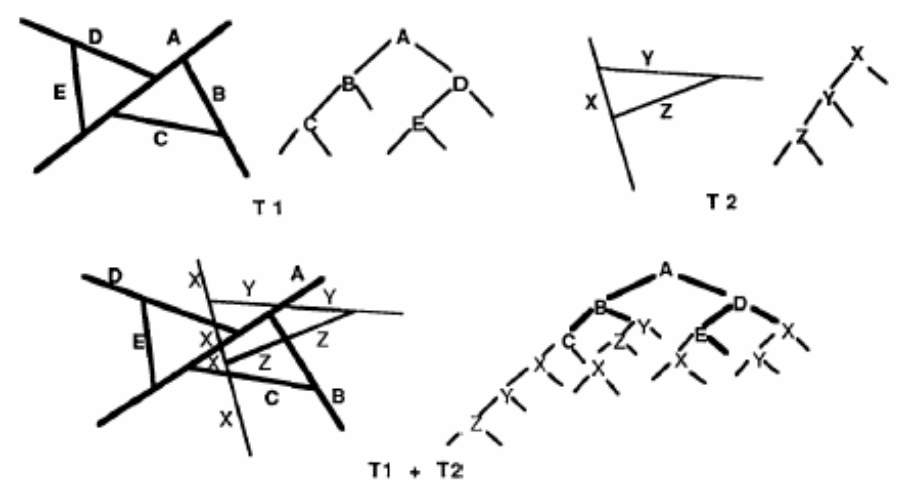

FIGURA 5.2
Fusionando dos árboles BSP [Naylor90].

El pseudocódigo de este procedimiento sería [Naylor90]:

ALGORITMO 5.1
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 **[Sección5.2.4]** y **partitionBSPTree** es la partición binaria de un árbol BSP **[Sección5.2.3]**.

A pesar de que este algoritmo muestra los principios básicos del algoritmo de fusión, existen algunos otros detalles importantes [Naylor90]. El primero de ellos proviene del hecho de que el algoritmo es completamente simétrico con respecto a sus dos operandos, así que en cada llamada recursiva se tiene la opción de elegir si se particiona el primer árbol por medio del segundo o viceversa. Segundo, también puede ser deseable realizar la operación de condensación ó reducción. Es decir, cuando los dos nodos hijos de un nodo interno resultan ser celdas del mismo tipo (ambas *in* o ambas *out*), no existe razón alguna para mantener dicha partición, así que el nodo interno se condensa en una sola celda del mismo tipo que el de sus hijos. Dada la suposición recursiva que los dos árboles a operar ya se encuentran condensados, esta operación puede realizarse a medida que la operación de fusión va progresando. En otras palabras, la operación de condensación en árboles BSP es homóloga a la condensación de OctTrees, excepto que a diferencia de éstos se puede realizar a medida que se realiza la fusión, y no como un procedimiento independiente y/o posterior.

5.2.3 Partición binaria de un árbol BSP

Dados un árbol BSP T y un particionador binario P definidos en la misma región del espacio, se necesitan obtener dos árboles, T^- y T^+, tal que $T^- = T \cap P^-$ y $T^+ = T \cap P^+$. Para computar estos dos árboles resultantes, se necesita usar nuevamente la noción de insertar una entidad geométrica en un árbol; en este caso, la entidad es un particionador binario. El proceso de inserción identificará qué regiones de T están completamente dentro de P^-, o completamente dentro de P^+, o bien son intersectadas por P (es importante notar que el procedimiento de inserción visita únicamente aquellas regiones que son intersectadas por P). Lograr esto requiere determinar las relaciones espaciales relativas entre dos particionadores binarios, y cuando se intersectan, dividir cada particionador binario usando la ecuación de soporte del otro [Naylor90].

El primer paso consiste en distinguir entre un nodo celda o un nodo interno. En el caso de encontrar un nodo interno, se deberá realizar la partición binaria de la entidad insertada. Particionar un nodo celda es trivial: sólo se requiere regresar dos copias de dicha celda. Sin embargo, para un nodo interno, el procedimiento no es tan inmediato. Lo primero consiste en realizar una bi-partición entre P y el particionador binario asociado al nodo, es decir, clasificar ambos particionadores uno respecto al otro de acuerdo a los casos estándares de partición binaria:

Localización: {negativa, positiva, ambas, coincidentes}

En **[Figura5.3]** se muestran cuatro de las siete posibles configuraciones geométricas (negativa/positiva, positiva/positiva y coincidentes-paralelas no son mostradas ya que tienen la misma geometría pero con una de las normales invertidas).

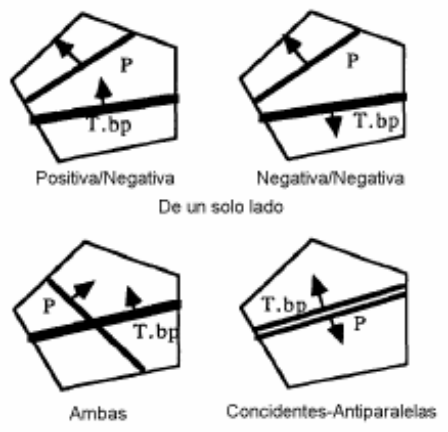

FIGURA 5.3
Relaciones espaciales entre dos particionadores binarios [Naylor90] (Traducción).

Aunque cada uno de los sietes casos se debe tratar por separado, todos ellos comparten la premisa básica de que cualquier subárbol que contenga al particionador insertado deberá ser particionado recursivamente, mientras que cualquiera que no lo contenga no necesitará ninguna modificación. Así que, por ejemplo, en el caso donde la localización de P sea *negativa,* resultará en que T.neg tendrá que ser particionado pero no así T.pos, mientras que si la localización de P es *positiva,* resultará en la acción opuesta. Una localización clasificada como *ambas* requerirá la partición de ambos subárboles, mientras que una clasificada como *coincidentes* no requerirá ninguna partición. Las partes de los subárboles producidas por esta partición recursiva deben unirse para formar los dos árboles que serán los valores de retorno de la operación [Naylor90].

Para aclarar esto, en **[Figura5.4]** se ilustra lo que sucede en el caso clasificado como *ambas*, en el cual se producen cuatro subárboles, dos para cada uno de los hijos de T. Durante el proceso de inserción de P en el árbol, la actividad se observa principalmente en términos de los dos semiespacios producidos por el particionador de T: se construye $P^- = P \cap T.neg$ y $P^+ = P \cap T.pos$. En contraste, el resultado, que se forma después de cualquier partición recursiva requerida, se encuentra en términos de los semiespacios producidos por P: $T^+ = T \cap P.pos$ y $T^- = T \cap P.neg$. De esta forma, el valor de retorno T^- es formado por piezas provenientes de ambos de los subárboles originales de T:

$$T^-.neg := T.neg^-$$

$$T^-.pos := T.pos^-$$

$$T^-.bp := T.bp^-$$

y de manera similar para T^+.

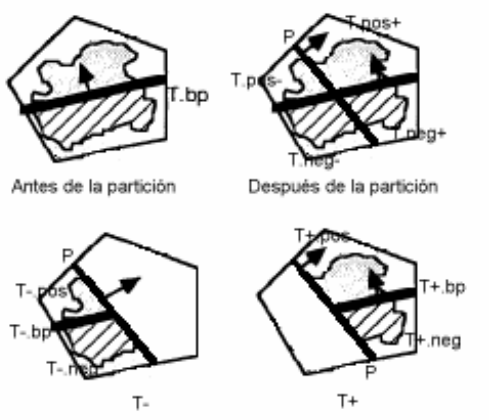

FIGURA 5.4
Partición de un árbol en el caso *ambas* [Naylor90] (Traducción).

Los casos en los que P se encuentra completamente en un lado del particionador binario de T se ilustran en **[Figura5.5]**. Existen cuatro instancias de este caso que se

obtienen invirtiendo las normales; sólo una es mostrada en la imagen. Para este caso, el particionador binario de T y T.neg permanecen intactos, mientras que T.pos sí es particionado. Los valores de retorno serían:

$$T^{-}.neg := T.neg$$

$$T^{-}.pos := T.pos^{-}$$

$$T^{-}.bp = T.bp$$

$$T^{+} := T.pos^{+}$$

mientras que para las otras tres instancias se tendrían asignaciones análogas.

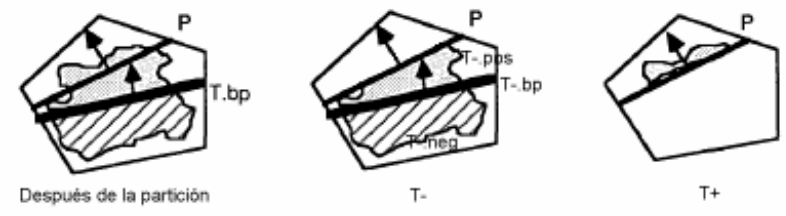

FIGURA 5.5
Partición de un árbol en el caso *positiva* [Naylor90] (Traducción).

Por último, para los casos *coincidentes*, no se requiere partición alguna, y el resultado se obtiene simplemente seleccionando los subárboles apropiados. Si las normales son paralelas:

$$T^{-} := T.neg$$

$$T^{+} := T.pos$$

y si no son paralelas:

$$T^{+} := T.neg$$

$$T^{-} := T.pos$$

Es importante mencionar que cualquier nuevo árbol formado debe tener la operación de condensación aplicada. Aunque no es necesaria para obtener resultados correctos, puede tener un impacto significativo en desempeño. En **[Figura5.6]** se observan dos objetos complejos. Si T2 se inserta en T1, entonces T2 será particionado por *X*, luego por *Y*, y por último por *Z*. En este momento, el fragmento de T2 dentro de T1 sería simplificado y condensado en una sola celda *out*, por lo que la operación de fusión estaría completa [Naylor90]. Cabe agregar una última observación: para fusionar T1 con T2, puede insertarse T2 en T1, lo cual conlleva la aparente paradoja de insertar T1 en T2 (específicamente, el particionador binario de T1), pero una pieza a la vez [Naylor90].

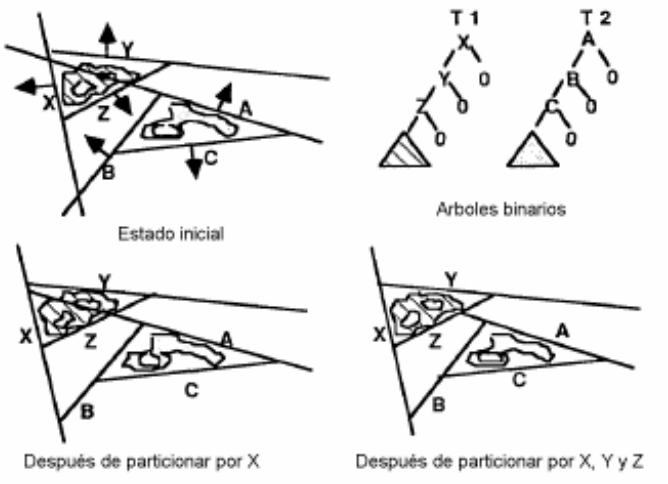

FIGURA 5.6
Efecto de condensar durante la partición [Naylor90] (Traducción).

5.2.4 Intersección, unión y diferencia

Una vez que el mecanismo para fusionar particiones espaciales está definido, realizar operaciones Booleanas entre poliedros definidos mediante el modelo BSP es una

cuestión relativamente sencilla. El proceso de fusión continúa recursivamente hasta que uno de los dos operandos en una celda, punto en el cual debe utilizarse un procedimiento para fusionar los atributos de la celda con aquellos de otro árbol arbitrario (que podría ser una celda también). Para operaciones Booleanas, esto implica seleccionar simplemente ya sea la celda o el árbol, posiblemente complementado **[Sección5.2.1]**. El pseudocódigo para realizar esto sería el siguiente:

ALGORITMO 5.2
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 de un árbol BSP **[Sección5.2.1]**.

5.2.5 Reducción de árboles BSP

Una vez que un árbol BSP ha sido construido como resultado de una operación Booleana, puede ser posible reducir el árbol eliminando ciertos nodos sin cambiar en realidad el objeto representado. Existen dos casos en los que esta reducción es posible. El

primero de ellos fue descrito en **[Sección5.2.2]**, y, como ya se mencionó, ocurre cuando ambos hijos de un nodo interno son celdas del mismo valor (nodo *z* en **[Figura5.7]**), simplificándose mediante la sustitución del mencionado nodo interno con una celda del mismo valor de sus hijos, procedimiento que puede realizarse como parte misma de la construcción del árbol resultado de la operación Booleana [Thibault87], [Naylor90].

Como segundo caso, una vez reconstruidas las fronteras de un árbol BSP según el procedimiento descrito en **[Sección4.2.1]**, también puede removerse todo nodo que tenga un hijo tipo celda, y, además, ninguna parte de la frontera del objeto es reconstruida a partir su subplano (nodo *u* en **[Figura5.7]**). Ya que el subplano del nodo no contribuye a una diferenciación real del espacio, este nodo puede ser reemplazado directamente por aquél de sus hijos que no sea una celda (nodo *w* en **[Figura5.7]**). Este procedimiento puede aplicarse una vez generado el árbol salida de una operación Booleana [Thibault87].

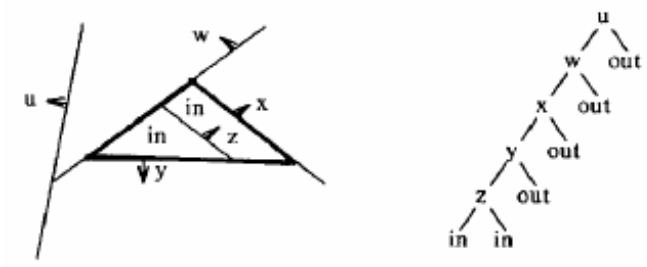

FIGURA 5.7
Reducción de árboles BSP. Los nodos *u* y *z* pueden eliminarse. [Thibault87].

5.3 Operaciones Booleanas entre BSP-OctTrees

Los algoritmos requeridos para la realización de operaciones Booleanas entre BSP-OctTrees están basados directamente en los utilizados en el modelo de OctTrees clásicos **[Sección2.4.4]**. De hecho, los únicos cambios requeridos aparecen cuando uno de los nodos a operar es un nodo BSP, en cuyo caso se usan exactamente los mismos algoritmos utilizados para el modelo BSP **[Sección5.2]**.

5.3.1 Complemento

Para obtener el complemento de un BSP-OctTree se debe realizar un recorrido por el árbol según los siguientes criterios [Navazo86], [Argüelles00]:

- Los nodos Blancos se transforman en Negros y los Negros en Blancos.
- Los nodos de tipo Gris permanecen inalterados.
- Los nodos BSP se complementan utilizando el algoritmo descrito en **[Sección5.2.1]**. Para ello, también es necesario modificar las orientaciones de los planos de soporte, que como ya se mencionó, se realiza cambiando los signos de los coeficientes de cada una de las ecuaciones en la tabla auxiliar de la codificación del objeto. Si el nodo BSP representará un árbol nulo (es decir, un nodo Gris de Mínima Resolución), éste permanece inalterado.

El algoritmo consiste de un procedimiento principal que en un primer paso copia la tabla auxiliar de las ecuaciones de soporte, pero cambiando los signos de todos los coeficientes de las mismas. Después de ello, se llama al procedimiento principal que construye el nuevo árbol a partir de la codificación DF del objeto original:

ALGORITMO 5.3
Complemento de BSP-OctTrees.

```
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 los vectores *ax*, *ay*, *az* son los mismos que los mencionados en **[Sección3.3.3]**.

5.3.2 Intersección

Intersectar dos BSP-OctTrees también es simple. El procedimiento es una variante del algoritmo del modelo clásico **[Sección2.4.4]**. Se examinan los nodos de cada árbol una

vez, avanzando en cada árbol de manera sincronizada y simultánea. Así pues, si en cierto momento el siguiente nodo de cada árbol es 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. Si los dos árboles tienen nodos Grises, entonces ambos recorridos continúan de manera recursiva [Ayala85], [Argüelles00]. En otras palabras [Navazo86], [Argüelles00], para la intersección de dos nodos clásicos se siguen los criterios descritos en **[Tabla2.2]**. En cualquier otro caso:

- La intersección entre dos nodos BSP requiere únicamente de operar los árboles BSP asociados a ambos nodos mediante los algoritmos descritos en **[Sección5.2]**. El árbol BSP resultante de dicha intersección formará un nuevo nodo BSP como salida, en caso de que dicho árbol describa un poliedro que cumpla con la restricción descrita en **[Sección3.3.3]**. En caso contrario, el árbol BSP resultante deberá ser utilizado para reconstruir los datos requeridos del modelo de fronteras descritos en **[Sección3.3.1]**, para después proceder a subdividir recursivamente este modelo y obtener un pequeño árbol BSP-OctTree, utilizando el algoritmo descrito en **[Sección3.3.3]**, que será la salida de la operación.
- 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 tamaño diferente.

El algoritmo consiste de un procedimiento principal que en un primer paso copia las tablas auxiliares de las ecuaciones de soporte de los dos árboles, combinándolas en una

sola tabla en el árbol resultado (simplemente “pegando” la segunda tabla al final de la primera). Una consecuencia importante de este paso es que al ir generando los nodos BSP del árbol resultado utilizando **[Tabla2.2]** los apuntadores a las ecuaciones de soporte en la tabla auxiliar contenidos en estos nodos (si los hay) no se copiarán exactamente de los nodos originales que intervinieron en la formación del nuevo nodo (como **sí** se hace en la operación complemento), sino que tendrá que verificarse primero si el apuntador pertenece al primer o segundo árbol, ya que los apuntadores que corresponden al primer árbol no tendrán alteración alguna, pero los apuntadores del segundo árbol deben ser modificados por un *offset*, equivalente al número de posiciones que tuvo que ser recorrida la segunda tabla al ser “pegada” al final de la primera [Argüelles00]. Como segundo paso, se llama al procedimiento principal que construye el nuevo árbol a partir de la codificación DF de los dos objetos originales:

ALGORITMO 5.4
Intersección de BSP-OctTrees.

```
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
   endif
  endfor
endprocedure
```

donde los vectores *ax*, *ay*, *az* son los mismos que los mencionados en **[Sección3.3.3]**. El procedimiento **mergeBSPTrees** es el descrito en **[Sección5.2.2]**. El procedimiento **buildOctTree** es el descrito en **[Sección3.3.3]**. El procedimiento **intersectGreyBSP** es descrito a continuación.

Como ya se mencionó, si en cierto momento el siguiente nodo de un árbol es un nodo Gris, mientras que en el otro árbol el siguiente nodo es 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. Cuando esto sucede, deben seguirse los siguientes criterios además de los de la ya mencionada **[Tabla2.2]**:

- 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, usando el mismo algoritmo descrito en **[Sección5.2.2]**, para después operar el resultado directamente con el nodo BSP menor. El resultado de esta segunda intersección debe agregarse como un nodo BSP en el árbol de salida si cumple con la restricción de **[Sección3.3.3]**, o subdividirse recursivamente en caso contrario.
- 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:

ALGORITMO 5.5

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
   endif
  endfor
endprocedure
```

donde los vectores *ax*, *ay*, *az* son los mismos que los mencionados en **[Sección3.3.3]**.

5.3.3 Unión y diferencia

Como se mencionó en **[Sección2.4.4]**, las operaciones de unión y diferencia en el modelo de OctTrees clásicos pueden realizarse aplicando ciertos cambios menores a los criterios establecidos por **[Tabla2.2]**. Aunque estos cambios se detallan con más precisión en [Navazo86], [Argüelles00], también se mencionó que estas operaciones pueden definirse utilizando únicamente las operaciones de intersección y complemento **[Sección2.4.4]**. Ahora bien, como se explica en **[Sección5.3.2]**, para extender los algoritmos de operaciones Booleanas del modelo clásico a los BSP-OctTrees únicamente es necesario definir el mecanismo para procesar los nodos BSP, lo cual puede hacerse directamente con los algoritmos descritos en **[Sección5.2]**, los cuales, además, ya no necesitan de ningún cambio para soportar las operaciones de unión o diferencia. En otras palabras, las operaciones de unión y diferencia en el modelo de los BSP-OctTrees pueden definirse ya sea modificando

la **[Tabla2.2]** 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 como ya fueron definidas.

5.4 Ventajas y desventajas

Las operaciones Booleanas en el modelo BSP-OctTrees son, nuevamente, una gran simplificación a sus contrapartes en el modelo de los PM-OctTrees. Dado que sólo existe un único tipo de nodo extendido, los nodos BSP, la extensión a partir de los algoritmos del modelo clásico, que son de orden lineal, es relativamente sencilla. Y, dado que los nodos BSP pueden utilizar de forma directa los algoritmos del modelo BSP, que son de orden cuadrático, el modelo propuesto no solo utiliza algoritmos de una muy baja complejidad, sino que también son altamente robustos.

Ya que el número de nodos de un árbol BSP-OctTree es mucho menor que su equivalente en el modelo clásico, el tiempo de procesamiento real termina siendo bastante menor a pesar del incremento de complejidad. Y lo mismo sucede al comparar con el modelo BSP, ya que los nodos BSP en el modelo propuesto describen, generalmente, árboles mucho más sencillos que los que se requerirían para objetos codificados directamente en BSPs. Sin embargo, para objetos sencillos ciertamente el tiempo de procesamiento será menor tanto en el modelo clásico como en los BSPs.