Capítulo 5.- Árboles BSP

La estructura de árboles BSP fue la utilizada para la implementación del navegador en este trabajo debido a que permite un desplegado muy rápido de los objetos (lineal con respecto al número de polígonos) independientemente de la posición del observador. En este capítulo se describirá detalladamente esta estructura de datos y la manera de generarla, utilizando y comparando diversos métodos. También se presentará el mecanismo para el desplegado del escenario virtual en pantalla.

5.1 Descripción

Los árboles de partición binaria del espacio (BSP) y sus algoritmos fueron desarrollados por Fuchs y Naylor y es un método extremadamente eficiente para calcular las relaciones de visibilidad entre un grupo estático de polígonos en 3D vistos desde un punto de vista arbitrario[FOLE90].

Esta estructura consiste en un árbol binario donde cada nodo contiene un polígono. El plano del polígono del nodo divide lógicamente el espacio en dos subespacios, siendo el frontal aquel que se encuentra del lado hacia donde apunta la normal del plano y el subespacio trasero el otro. Además del polígono, cada nodo contiene un apuntador a cada uno de estos dos subespacios, los cuáles a su vez son subárboles.

Esta representación de escenas no es única, es decir, un mismo escenario puede ser representado utilizando diferentes árboles BSP, unos con mayor número de nodos que otros o más o menos balanceados, lo que se refleja en que unos sean más rápidos que otros para desplegar objetos.

5.2 Generación del árbol BSP

El árbol BSP que representa una escena es generado una vez que se tiene el conjunto de polígonos que conforman la escena, ya sea en un arreglo, lista ligada u otra estructura de datos semejante.

El árbol BSP es generado eligiendo cualquier polígono como polígono raíz. Este polígono es utilizado para dividir el espacio en dos subespacios. Uno contiene los polígonos que se encuentra enfrente del polígono raíz, relativo a su normal, y el otro subespacio los polígonos detrás de él. Si algún polígono pertenece a ambos subespacios es dividido por el plano en dos partes y cada uno de estos polígonos es introducido en su respectivo subespacio. Si un polígono está en el mismo plano del polígono raíz puede colocarse en cualquier subespacio. Un polígono del subespacio frontal y otro del trasero se vuelven el hijo frontal e hijo trasero del nodo y cada hijo es recursivamente usado para dividir su subespacio de la misma manera. El algoritmo continúa hasta que cada nodo contiene un solo polígono. A continuación se muestra el pseudo-código para generar un árbol BSP[FOLE90]:

```
type
   BSP_tree = record
      root : polygon;
      backChild, frontChild : BSP_tree;
   end;
function BSP_makeTree(polyList : listOfPolygons) : BSP_tree;
var
   root : polygon;
   backList, frontList : listOfPolygons;
   p, backPart, frontPart : polygon;{se asume que cada polígono es
convexo}
begin
   if polyList is empty then
      BSP_makeTree:=nil;
   else
      begin
         root:=BSP_selectAndRemovePoly(polyList);
         backList:=nil;
         frontList:=nil;
         for each remaining polygon p in polyList
            begin
               if polygon p in front or inside of root then
                  BSP_addToList (p,frontList);
               else if polygon p in back of rootv then
                  BSP_addToList (p,backList);
               else{polígono debe ser dividido}
                  BSP_splitPoly (p, root, frontPart, backPart);
                  BSP_addToList (frontPart,frontList);
                  BSP_addToList (backPart,backList);
               end
            end;
        BSP_makeTree:=BSP_combineTree
(BSP_makeTree(frontList),root,BSP_makeTree(backList));
      end
end;
```

Debido a que durante la creación del árbol BSP pueden ocurrir divisiones de polígonos, en la mayoría de las escenas el número de polígonos después de la creación del árbol BSP es mayor que el número de polígonos originales. La elección del polígono para particionar cada subespacio repercute drásticamente en el número de polígonos divididos. La figura 5.1 muestra el proceso de creación de un árbol BSP y un árbol alterno eligiendo otro polígono para particionar el espacio.

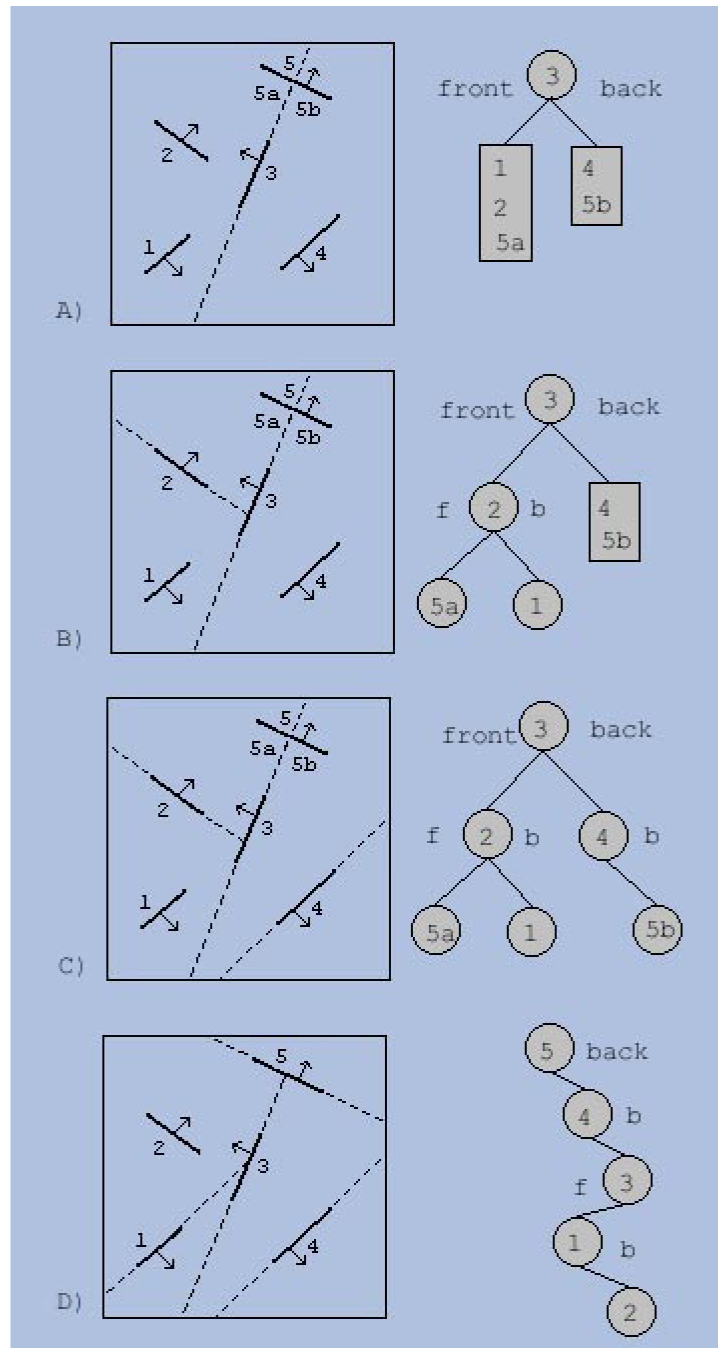

Fig. 5.1.- Proceso de creado de un árbol BSP. A)Vista de la escena con el árbol BSP antes de la recursión con polígono 3 como raíz. B)Después de construir el subárbol izquierdo. C)Árbol completo D)Árbol alterno con el polígono 5 como raíz.

De todos los árboles BSP que representan la misma escena, aquel con el menor número de nodos será el que permita un desplegado más rápido del mismo, debido a que para cada polígono que compone el escenario, se deben realizar operaciones para

determinar la posición del observador respecto al plano de ese polígono y llamar a una función para pintarlo, por lo que la disminución del número de polígonos reduce el número de cálculos. En caso de que dos o más árboles contengan el mismo número de nodos, aquel mejor balanceado será más rápido, pues será menor la profundidad de recursión. Este árbol que permite el desplegado más rápido de la escena es llamado el árbol óptimo, sin embargo, el algoritmo para encontrarlo es demasiado tardado pues el número de combinaciones de árboles es muy grande (ver sección 5.2.4)..

5.2.1 Posición de un polígono respecto a un plano

Para determinar si un polígono Q se encuentra enfrente, detrás o a ambos lados del plano de otro polígono P es necesario evaluar cada vértice de Q para determinar el lado donde se encuentra utilizando el método explicado en la sección 4.1.4. En algunos casos es posible que por errores de redondeo, un vértice sea evaluado como si estuviera en un lado incorrecto. Este caso es problemático cuando el polígono está del lado contrario de donde este vértice fue evaluado debido a que se producirá una división del polígono, generando uno demasiado pequeño (ver figura 5.2). Para evitar esto puede utilizarse un factor de error de redondeo durante la evaluación de la posición del vértice respecto al plano.

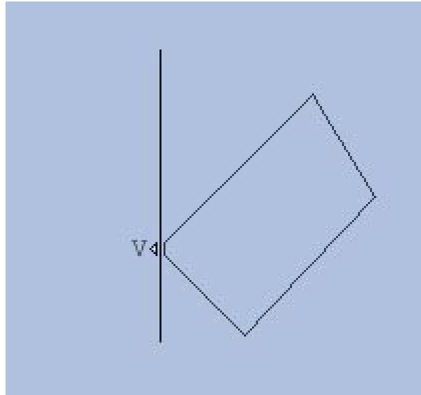

Fig. 5.2.- Por errores de redondeo el vértice *V* puede ser evaluado del lado contrario al que en realidad se encuentra, provocando la división del polígono.

El siguiente pseudo-código realiza la evaluación del lado donde se encuentra un polígono respecto a un plano tomando en cuenta un factor para el error de redondeo:

```
function evaluaPolygono(plano,poligono: polygon):integer
var
       value:double;
       i,result:integer;
       A,B,C,D:double;
begin
    result := INSIDE;
    {obtener ecuación del plano}
    With plano begin
         A := .normalx;
         B := .normaly;
         C := .normalz;
        D:=(A * allVerticesx(.vertices[0]) +
            B * allVerticesy(.vertices[0]) +
            C * allVerticesz(.vertices[0]));
    end;
    With poligono begin
         {determinar el lado de cada punto del poligono}
         For i = 0 To .points - 1 begin
            value = A * allVerticesx(.vertices(i)) +
                     B * allVerticesy(.vertices(i)) +
                     C * allVerticesz(.vertices(i)) +
                     D;
            If (value < -EPSILON And result = INFRONT) Or (value >
                EPSILON And result = INBACK) Then begin
                result := SPLITED;
                exit for;
            end
            If value < -EPSILON Then
                result := INBACK;
            ElseIf value > EPSILON Then
                result := INFRONT;
        Next i
    end;
    evaluaPolygono := result;
end;
```

5.2.2 División de polígonos

Durante la creación del árbol BSP un polígono puede ser dividido por un plano en dos partes. Cada uno de estos dos polígonos tendrá una arista extra (aquella que es compartida por ambos polígonos). Además, si alguno de estos dos polígonos es de nuevo dividido, el número de aristas extras crecerá. Esto no presenta un problema si se va a dibujar sólo el área del polígono, sin embargo, si también van a ser pintadas las aristas, por ejemplo, para mostrar los objetos como modelos de alambre, estas aristas extras serán visualizadas. Compárense las figuras 5.3 A) y B); la primera muestra la imagen correcta al dibujar los polígonos y sus aristas, mientras que la segunda presenta la deficiencia de que las aristas extras creadas durante el proceso creación del árbol BSP al dividir polígonos, son mostradas.

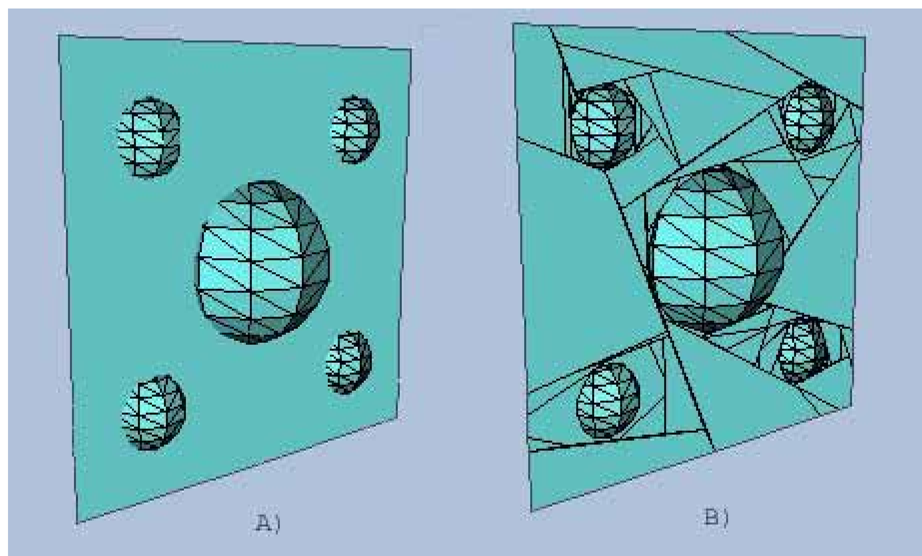

Fig. 5.3.- Imagen de esferas atravesadas por un polígono A)dibujando correctamente las aristas originales de los polígonos y B)dibujando las aristas originales y aristas extras creadas al dividir polígonos.

Para poder presentar correctamente las aristas de los polígonos divididos es necesario implementar un mecanismo que indique qué aristas son originales y qué aristas no lo son. Estas aristas extras son llamadas aristas virtuales y se puede utilizar una lista *a* del mismo tamaño que la lista de vértices del polígono, donde elementos en la misma posición de ambas listas se refieren al mismo vértice. Utilizando esta estructura, un valor 0 en la posición *i* de la lista *a* quiere decir que la arista que llega al vértice *i* del polígono es virtual y no debe ser pintada (a menos que se desee mostrar la fragmentación); un valor 1 indica que la arista es original.

La figura 5.4 muestra el ejemplo de un polígono dividido por un plano mostrando los vértices de cada polígonos y sus respectivos indicadores de aristas virtuales. Es importante destacar que una vez dividido un polígono en dos, si el área de éstos es dibujada se obtendrá la misma imagen que si el área del polígono original es dibujada.

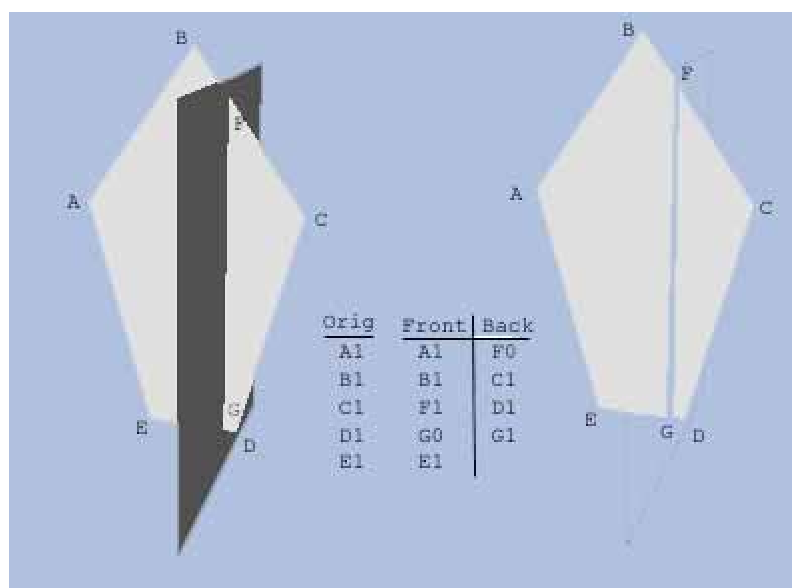

Fig. 5.4.- División de un polígono por un plano mostrando los indicadores de aristas virtuales antes y después de la división del polígono.

Los indicadores de aristas virtuales son inicializados en 1 para todos los vértices del polígono. El algoritmo consiste en recorrer cada uno de los vértices del polígono a dividirse determinando mediante el método visto en la sección 4.1.4 el lado en el que se encuentra el vértice respecto al plano de partición e insertándolo en el polígono correspondiente, junto con el indicador de arista virtual almacenado para ese vértice. Cuando se detecta que un vértice v_2 se encuentra del lado contrario del vértice anterior v_1 respecto al plano, se calcula la intersección de la arista v_1v_2 con el plano mediante el mecanismo explicado en la sección 4.1.5. El punto generado es añadido a ambos polígonos colocando un 0 en el indicador de arista virtual del vértice v_2. El siguiente pseudo-código realiza esta división (el procedimiento insertaVertice inserta el vértice en el polígono correspondiente de acuerdo al valor recibido como tercer parámetro):

```
ultimoLado := evaluaLadoDeVertice(vertice[1], plano);
InsertaVertice(vertice[1], virtual[1], ultimoLado);
lado1:=ultimoLado;{para comparar el lado del primer y ultimo vértice}
For k:=2 to numVertices begin
      lado := evaluaLadoDeVertice(vertice[k], plano);
      If lado<> ultimoLado begin
            inters:=calculaInterseccion(vertice[k],vertice[k-1],plano);
            InsertaVertice(inters, virtual[k], ultimoLado);
            InsertaVertice(inters, 0, lado);
            ultimoLado := lado;
      end;
      insertaVertice(vertice[k],virtual[k],lado);
end;
if lado1<>ultimoLado then begin
      inters:=calculaInterseccion(vertice[1],vertice[numVertices],plano);
      InsertaVertice(inters, 0, ultimoLado);
      InsertaVertice(inters, virtual[k], lado1);
end;
```

La función para calcular la intersección de una arista con el plano es:

```
Function calculaInterseccion(punto1,punto2:integer;
                             plano:polygon):punto3D
var
      vx,vy,vz,temp1,temp2:double
      I:punto3D;
begin
      vx := allVerticesx[punto2] – allVerticesx[punto1];
      vy := allVerticesy[punto2] – allVerticesy[punto1];
      vz := allVerticesz[punto2] – allVerticesz[punto1];
      With poligono begin
         {Temp1=A(x1-xo) + B(y1-yo) + C(z1-zo)}
         temp1 := .normalx * vx + .normaly * vy + .normalz * vz;
         {Temp2=Axo + Byo + Czo + D}
         temp2 := -(.normalx * allVerticesx[punto1] +
                    .normaly * allVerticesy[punto1] +
                    .normalz * allVerticesz[punto1] + D)
         I.x := allVerticesx[punto1] + temp2 * vx / temp1;
         I.y := allVerticesy[punto1] + temp2 * vy / temp1;
         I.z := allVerticesz[punto1] + temp2 * vz / temp1;
      end;
      calculaInterseccion := I;
end;
```

Durante la división de polígonos es posible realizar algunas reducciones de vértices innecesarios (aquellos cuya arista es demasiado pequeña como para tomarse en cuenta) generados principalmente por errores de redondeo (ver sección anterior) o por múltiples divisiones en el proceso de creación del árbol BSP. Para corregir estos casos basta con eliminar uno de dos vértices consecutivos cuya distancia sea menor que un cierto épsilon. Al realizar estas reducciones de vértices hay que tomar en cuenta que si el polígono es un triángulo, al eliminar un vértice quedarán sólo dos aristas, por lo que lo mejor es eliminar completamente el polígono.

5.2.3 Elección del polígono base

Como ya se mencionó, la elección del polígono raíz para cada subespacio es muy importante, ya que repercute en el número de polígonos divididos y, por lo tanto, en la velocidad de desplegado de la escena.

Existen varias técnicas para la elección de este polígono, unas más rápidas que otras. A continuación se listan varios tipos de decisiones para su elección:

+*El primero*.- bajo este enfoque se toma el primer polígono en la lista como polígono raíz. Este método es sumamente rápido pues no necesita realizar comparaciones con otros polígonos para la elección, sin embargo, el árbol generado en la mayoría de los casos es bastante malo comparado con otros métodos. Tiene la característica de elegir polígonos que pertenecen a los mismos objetos, debido a que estos polígonos se encuentran muy cerca unos de otros en la lista.

+*Uno al azar*.- elige un polígono de la lista al azar. Este método al igual que el anterior es muy rápido y más efectivo debido a que comúnmente toma polígonos de diferentes cuerpos.

+*El mejor de varios al azar*.- se elige algún número fijo de polígonos de la lista al azar. Cada uno de estos polígonos es evaluado contra todos los demás para determinar cuántos polígonos divide. Se elige el polígono que genere el menor número de divisiones. Si hay empate se puede elegir aquel que tenga la menor diferencia entre el número de polígonos enfrente de él y el número de polígonos detrás de él, para generar así el árbol más balanceado. Este método encuentra árboles BSP mucho mejores que los métodos anteriores, sin embargo aumenta considerablemente el tiempo en crearlo, debido al costoso proceso de evaluar los polígonos elegidos contra todos los demás.

+*El mejor de todos*.- utilizando el mismo mecanismo que el método anterior evalúa todos los polígonos de la lista para determinar cuál es el mejor. Este mecanismo genera árboles muy cercanos al óptimo en un tiempo de ejecución razonable.

El mecanismo de la elección del mejor polígono puede ilustrarse con el siguiente algoritmo:

```
function obtenElMejor(lista: listaPoligon; tam:integer) : integer
var
    i, num, numfrontback:integer;
    numdivide, mindivide, minfrontback, minindex:integer;
begin
    If tam = 1 Then
        obtenElMejor := 1;
    Else
        mindivide := NUMERO_MUY_GRANDE;
        minfrontback = NUMERO_MUY_GRANDE;
        For num = 1 To tam begin
            'contar cuantos poligonos divide
            numdivide = 0
            numfrontback = 0
            For i = 1 To tam begin
                If num <> i Then
                    Switch evaluaPolygono(lista(num), lista(i)) begin
                        INFRONT:
                            numfrontback := numfrontback + 1;
                        INBACK:
                            numfrontback := numfrontback - 1;
                        SPLITED:
                            numdivide := numdivide + 1;
                    end;
                    If numdivide > mindivide Then
                        Exit For;
                End If
            End;
            If numdivide < mindivide Or (numdivide = mindivide And
               Abs(minfrontback) - Abs(numfrontback) > 0) Then begin
                mindivide := numdivide;
                minfrontback := numfrontback;
                minindex := num;
            end;
        end;
        obtenElMejor := minindex;
    end;
end;
```

En la sección 7.2 se muestra un estudio detallado de varias pruebas de la creación del árbol BSP con varias escenas utilizando estos métodos, mostrando el tiempo tardado en generar el árbol y el número de polígonos en total.

5.2.4 Análisis de complejidad

La forma de encontrar el árbol óptimo es evaluando cada uno de los árboles posibles que representen la escena y eligiendo el que tenga el menor número de nodos y en caso de empate el más balanceado. El número total de combinaciones de árboles BSP de un mundo no puede determinarse exactamente (basado en el número de polígonos originales) ya que dependiendo del polígono raíz elegido para cada subárbol varios polígonos pueden ser divididos, incrementándose el número de polígonos y combinaciones. Además, el número máximo y mínimo de polígonos que pueden ser divididos depende de las características particulares de la escena.

Sin embargo es posible determinar una cota inferior del número de combinaciones evaluando el mejor caso. Éste ocurre en escenarios donde cualquier polígono que se elija como raíz no produce particiones y además el subespacio frontal de todos polígonos o el subespacio trasero de todos los polígonos esté vacío, de manera que el árbol generado es una lista de polígonos(Fig. 5.5). Sea n el número de polígonos originales que comprenden la escena, el número de combinaciones posibles de árboles BSP para este caso es $n!$. Por lo tanto, el algoritmo para encontrar el mejor árbol BSP es por lo menos $O(n!)$, tomando como referencia el número de árboles BSP evaluados.

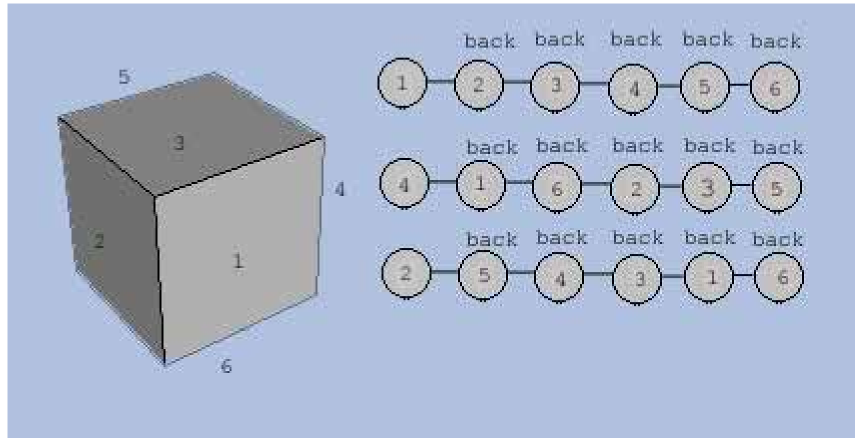

Fig. 5.5.- Izquierda)Escenario óptimo para la generación del árbol BSP.
Derecha)Árboles BSP que representa ese escenario.

Para el caso del cubo de la figura 5.5 *n* es igual a seis y el número de árboles BSP es 720. Para un escenario con una esfera con factor de poligonalización igual a 7 (70 polígonos), el número de combinaciones llega a $1.2\text{x}10^{100}$. Como la mayoría de las escenas consta de cientos de polígonos, obviamente evaluar todos sus árboles BSP es prácticamente imposible.

Los métodos descritos en la sección anterior obtienen un solo árbol BSP, eligiendo "adecuadamente" el polígono raíz para cada subárbol. La complejidad de estos algoritmos puede ser evaluada tomando en cuenta el número de comparaciones entre polígonos para determinar el polígonos raíz. En el caso de tomar el primer polígono o uno al azar no se realizan comparaciones, sin embargo en elegir el mejor o varios al azar esto es necesario.

Analizando el algoritmo de elegir el mejor polígono de la lista es sencillo determinar una cota superior para el algoritmo. El peor caso ocurre cuando la elección del mejor polígono raíz para cada subárbol no divide el espacio en dos, sino que todos los

demás polígonos quedan de un mismo lado, provocando que todos los polígonos restantes sean evaluados entre ellos, necesitando en cada subárbol m*(m-1) comparaciones, siendo m el número de polígonos de ese subárbol. Este caso ocurre cuando hay un solo cuerpo en el escenario, como en la figura 5.5. Ahora, debido a que en cada nivel del árbol un polígono es elegido y no es evaluado en los subárboles siguientes, el árbol tendrá una profundidad de n nodos, donde cada nivel tiene un polígono menos que el anterior. Entonces el número de polígonos evaluados es:

$$\sum_{i=1}^{i=n}\left(i\times\left(i-1\right)\right)=\sum_{i=1}^{i=n}i^{2}-\sum_{i=1}^{i=n}i=\frac{i^{2}\left(i^{2}+1\right)}{2}-\frac{i\left(i+1\right)}{2}=\frac{i^{4}-i}{2}$$

y el algoritmo para encontrar el árbol BSP eligiendo el mejor polígono de cada subespacio es a lo más O(n^4), siendo n el número de polígonos que forman la escena.

5.3 Desplegado del árbol BSP

Los árboles BSP pertenecen a la categoría de algoritmos de lista de prioridad (ver sección 3.1.4) en los que el objetivo es tener los polígonos en una estructura conociendo qué polígonos están más alejados de otros.. La idea de este algoritmo es que un polígono puede ser pintado completamente si:

1.- Se asegura que ningún polígono lo obscurece y no hay interpolaciones cíclicas con otros polígonos.

2.- Todos los polígono que se encuentra detrás de él (relativo al punto de vista del observador) ya fueron pintados.

El primer elemento se logra durante la creación del árbol, dividiendo los polígonos que intersectan el plano del polígono elegido en un subespacio, de manera que las interpolaciones cíclicas son eliminadas. El segundo elemento es tomado en cuenta dibujando primero todos los polígonos que están detrás de un polígono, luego él mismo y finalmente todos los que se encuentran enfrente de él o en sentido contrario, dependiendo si el observador se encuentra enfrente o detrás del polígono.

Para desplegar el árbol se comienza con el nodo raíz. Primero es necesario determinar si el observador está enfrente de este polígono o detrás (relativo a la normal del polígono). Esto se puede lograr obteniendo el signo de sustituir la posición (x, y, z) del observador en la ecuación del plano del polígono $Ax + By + Cz + D = 0$. Ahora, si el observador se encuentra enfrente del polígono, primero se llama a pintar recursivamente al *back child*, luego se pinta ese polígono y finalmente se pinta el *front child*. Si al evaluar la posición del observador respecto a un polígono resulta que se encuentra detrás de él, el proceso se invierte: primero se dibuja el *front child*, luego el polígono y finalmente el *back child*. Se puede dar el caso en que el observador se encuentre en el plano del polígono actualmente evaluado, donde es indistinto qué subespacio se dibuje primero. Además, para cada polígono, si se determina que no es visible al observador utilizando *back-face culling* (ver sección 3.1), no es necesario pintarlo. La figura 5.6 muestra el ordenamiento de polígonos al pintar un árbol BSP desde dos puntos de vista diferentes.

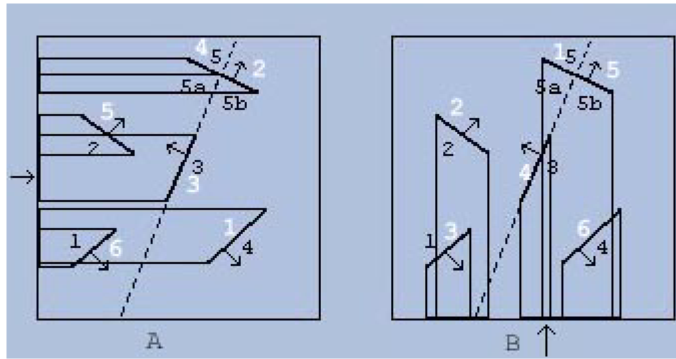

Fig. 5.6 Dos proyecciones del desplegado del árbol BSP estando el observador A)a la izquierda y B)abajo. Las líneas proyectados son mostradas más gruesas. Los números blancos indican el orden en que se dibujan los polígonos.

El pseudo-código para el desplegado del árbol BSP en pantalla es el siguiente[FOLE90]:

```
procedure BSP_displayTree (tree : BSP_tree);
begin
   if tree is not empty then
      if viewer is in front of root then
         begin
            {display back-child, root, and front child.}
            BSP_displayTree( tree.backChild);
            displayPolygon (tree.root);
            BSP_displayTree( tree.frontChild);
         end
      else
         begin
            {display front-child, root, and back child.}
            BSP_displayTree( tree.frontChild);
            displayPolygon (tree.root); {only if back-face culling
                                         not desired}
            BSP_displayTree( tree.backChild);
         end
end;
```