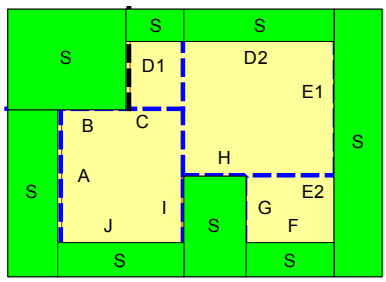

Binary Space Partitioning

parte IV

Escribiendo el árbol Bsp en un archivo

Autor: Diego G. Ruiz

Versión: 1.0 (draft)

Tabla de Contenidos

Binary Space Partitioning (BSP)

parte IV

Introducción

El proceso de generación de un mapa BSP puede ser extenso, más aún puede serlo la generación de información complementaria al mismo relacionada a la visualización del mapa. Por lo tanto, es conveniente que este mapa no se genere dinámicamente durante la ejecución del juego, sino que se realice en una etapa previa y se almacene en un formato adecuado.

Claro, que esto acarrea una de las principales desventajas de los árboles BSP y es que son **estáticos**, es decir, el orden de los polígonos no puede variar durante el juego (esto complica la implementación de "geometría desformable") .

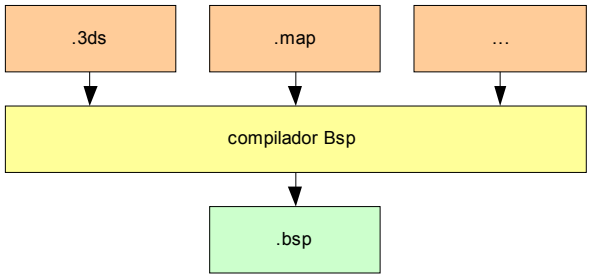

Figura 1. El árbol bsp es generado y almacenado en un archivo

El árbol bsp se puede generar a partir de un archivo que almacene información de geometría como el 3ds, map, o cualquier otro.

Luego, el motor no tendrá que saber de generaciones de árboles bsp sino de cómo interpretarlo.

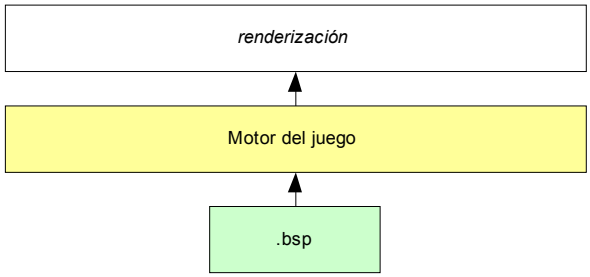

Figura 2. El archivo donde se almacenó el árbol bsp es interpretado por el motor

El formato de archivo

En principio, nuestro archivo poseerá una cabecera con información general relacionada al archivo, veamos:

int id: Una constante que identifique nuestro tipo de archivo (Ej: 'ABSP').

int ver: Una versión especificada para nuestra definición de archivo de mapa (de este modo el programa que lo cargue sabrá como interpretar la información que esté dentro de él, si es que soporta la versión en cuestión).

DirEntries dir[N]: Directorio para acceder a las distintas unidades de información de nuestro archivo (N será la cantidad total de lumps distinto que definamos para nuestro archivo, por el momento serán 3).

La identificación y la versión son propiedades que ya hemos vistos en muchos formatos de archivos pero ¿qué es DirEntries? Nuestro archivo poseerá distintas unidades de información como ser un nodo bsp, por ejemplo, que está compuesto del siguiente modo:

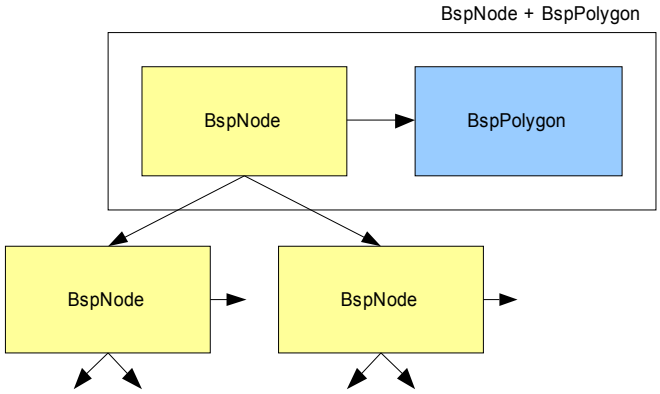

Figura 3. Un nodo de nuestro árbol bsp

Todo nodo de nuestro árbol poseerá tres posibles punteros: un puntero al nodo de la geometría que se encuentra detrás, un puntero al nodo de la geometría que se encuentra delante y un puntero al polígono utilizado como splitter en el nivel correspondiente al nodo. Claro que alguno de estos punteros puede ser NULL, por ser el nodo una hoja, siendo este el caso la unidad de información la clasificaremos como otra diferente.

El tipo de dato **DirEntries** será una estructura con las siguientes propiedades:

int offset: Cantidad de bytes desde el inicio del archivo para alcanzar el lump correspondiente.

int length: Tamaño en bytes del tipo de lump. Siempre múltiplo de 4.

Entonces repasemos, nuestras unidades de información (que también llamaremos lumps) hasta el momento son las siguientes:

Índice	Lump	Descripción
0	NodoBsp	Especifica información de un nodo bsp.
1	NodoHojaBsp	Especifica información de la hoja de un nodo bsp.
2	Polígono	Polígono utilizado por el nodo bsp para realizar el split.
3	Material	Especificación de material y textura.

Tabla 1. Listado parcial de lumps.

Por lo tanto, la estructura general de nuestro archivo será la siguiente:

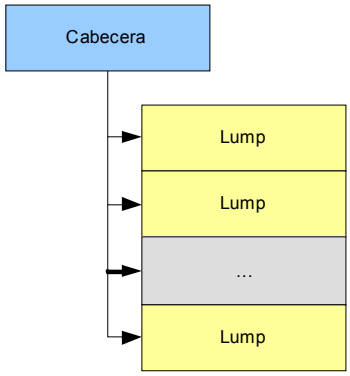

Figura 4. La estructura de nuestro archivo

Definición del lump NodoBsp y NodoHojaBsp

Recordemos como está compuesto el nodo bsp que utilizamos para generar el mapa bsp:

```
struct BspNode
{
	BspPolygon * m_pSplitter;
	BspNode * m_pFront;
	BspNode * m_pBack;
	bool m_bIsLeaf;
	bool m_bIsSolid;
};
```

Pero en el archivo no tiene sentido almacenar punteros, ya que son direcciones de memoria otorgadas por el sistema en un determinado contexto de ejecución. En su lugar, debido a que todos los nodos son almacenados dentro del archivo de modo contigüo le daremos a cada nodo un índice, lo mismo que con los polígonos y reemplazaremos la información de dirección de memoria por dicho índice.

Por lo tanto nuestro nodo especificado en el archivo quedará compuesto por:

```
int splitter: Índice del polígono utilizado como splitter.
int[2] children: Índices de los dos nodos hijos del nodo actual.
```

Claramente se puede entender que en un nodo no-hoja no es necesario almacenar la información **m_bIsLeaf** ni **m_bIsSolid**.

Por el momento nuestro nodo hoja poseerá la siguiente información:

bool solid: Indica si la hoja es un sólido.

Claro, que luego deberemos agregar información relacionada a los PVS.

Definición del polígono

La estructura del polígono utilizado por era la siguiente:

```
struct BspPolygon
{
	TexNormVertex m_verts[10];
	short m_idxs[24];
	Vector3 m_v3Normal;
	int m_iNumberOfVerts;
	int m_iNumberOfIdxs;
	BspPolygon * m_pNext;

	// Material relacionado al polígono
	int m_iMaterialIdx;
};
```

Naturalmente, aquí tampoco aquí almacenaremos punteros por lo tanto nuestro polígono será almacenado del siguiente modo:

TexNormVertex m_verts[10]: Array de vértices del tipo posición + normal + coordenadas de textura.

short m_idxs[24]: Array de 24 shorts especificando índices.

int numVerts: Número de vértices.

int numIdxs: Número de índices.

int materialId: Índice del material relacionado.

Definición de un material

El material referenciado por un polígono poseerá las siguientes propiedades:

int dif[4]: Componente diffuse, array de 4 enteros (RGBA).

int amb[4]: Componente ambiente, array de 4 enteros (RGBA).

int spe[4]: Componente especular, array de 4 enteros (RGBA).

char textureName[64]: Nombre del mapa de bits relacionado al material.

Hasta aquí acabamos con lo que será nuestra definición de formato de archivos bsp versión 1. Más adelante deberemos hacer algunos cambios, ya que se agregará información y cambiaremos un poco la definición de un nodo bsp y la hoja bsp, pero la estructura general del archivo será la misma.