

## 2. Mapas de terreno

Aunque el objetivo de este tutorial es asentar unos conocimientos básicos sobre programación de videojuegos con Bullet 3D, conviene también tener unas pequeñas nociones sobre diseño de elementos ambientales. En este tema trataremos los mapas de terreno (Height maps), que son una forma sencilla y eficiente de generar escenarios a cielo abierto, y aprovecharemos para introducir algunos conceptos básicos sobre texturas, nieblas e iluminación que nos serán útiles para el proceso de desarrollo de nuestro videojuego.

### 2.1. Obtener mapas de terreno

Un mapa de terreno es una simple imagen plana en cualquiera de los formatos de textura soportados por Bullet 3D (Ver formatos en la lista de funciones). Se trata de imágenes en escala de grises en las que las zonas más oscuras corresponden a zonas profundas del escenario mientras que las zonas más claras corresponden a las zonas más elevadas.

Podemos generar aleatoriamente y editar mapas de terreno con la herramienta Terrain Editor incluida en el Bennupack 1.8. Esta herramienta permite exportar los mapas a formato .bmp, aunque es recomendable que posteriormente los conviertas a .png, que será el formato de imagen que utilizaremos durante todo el tutorial por su perfecto equilibrio entre calidad y tamaño.

También puedes optar por seleccionar entre uno de los mapas de terreno del directorio de recursos (<http://trinit.es/DescargaDirecta/RecursosVideojuegos/>), ya que se encuentran perfectamente catalogados por su topología.

Para nuestra primera prueba optaremos por una solución distinta. Dado que necesitaremos también una textura apropiada para nuestro terreno, seleccionaremos una textura de terreno y la convertiremos a escala de grises, así tendremos por un lado el mapa de terreno y por otro una textura que encajará sobre él a la perfección. Si seleccionas con cuidado la textura puedes conseguir un efecto asombroso.

Como ya anticipamos en el anterior tema, es importante ser ordenados, así que guardaremos tanto el mapa como la textura en el subdirectorio /map de nuestro videojuego, el nombre del fichero del mapa será 001.png y el nombre del fichero con su textura será t001.png. Mapas sucesivos podrán nombrarse con números sucesivos.

### 2.2 Cargar un mapa de terreno con textura

La carga del mapa de terreno sustituirá a las instrucciones de carga de un mapa de Quake III del videojuego de ejemplo del tema anterior. Recuerda que con `M8E_ADDZIPFILE ( )` descomprimamos el fichero con el mapa de Quake y con `M8E_LOADMODELEX ( )` cargábamos el mapa y almacenábamos su identificador en la variable global mapa. Eliminaremos estas instrucciones y pondremos en su lugar las siguientes.

La instrucción que carga un terreno es `M8E_LOADTERRAIN ( )`, y recibe como parámetro una string con la ruta del mapa a cargar, en nuestro caso "map/001.png". La instrucción genera un identificador del modelo de terreno cargado, y guardaremos ese identificador en la variable global `mapa`, por lo que quedaría:

```
mapa = M8E_LOADTERRAIN ( "map/001.png" );
```

Para cargar la textura del terreno necesitaremos una nueva variable donde guardar su identificador, también se trata de un entero en el caso de las texturas. Pero en este caso no utilizaremos una variable global, ya que se trata de un identificador que no tiene por qué ser accedido por otros procesos, en su lugar utilizaremos la variable predefinida `graph`. Se trata de una variable predefinida en todo proceso de `BennuGD` que no necesitamos declarar y podemos utilizar directamente, esta variable existe ya que en `BennuGD` se asume que la mayoría de los procesos usarán un gráfico.

La instrucción que carga una textura es `M8E_LOADTEXTURE ( )`, nuevamente recibe como parámetro una string con la ruta de la textura a cargar y genera un identificador de la textura cargada. En nuestro caso quedaría:

```
graph = M8E_LOADTEXTURE ( "map/t001.png" );
```

Por último, observa que hasta ahora no hemos hecho más que cargar un modelo y una textura, todavía no hemos establecido una relación entre ambos. Para ello hay que utilizar la instrucción `M8E_LOADTEXMODEL ( )` cuya única finalidad es aplicar una textura a un modelo. El primer parámetro de esta función debe ser el identificador del modelo a texturizar, que viene dado por la variable `mapa` en nuestro caso, y el segundo parámetro es el identificador de la textura, que viene dado por la variable `graph`. Quedaría:

```
M8E_LOADTEXMODEL ( mapa , graph );
```

Esta última instrucción aplica la textura pero no genera ningún identificador.

### 2.3 Aplicar iluminación

Habrás observado que, excepto el mapa de Quake que cargaba el videojuego de ejemplo del Tema 1, todos los modelos cargados se ven de color negro y es inapreciable incluso la textura que les aplicamos.

Esto se debe a que es necesaria una fuente de luz para poder apreciar los colores. El caso del mapa de Quake es una excepción porque fue cargado con `M8E_LOADMODELEX ( )` una función que optimiza la carga de modelos de alta densidad poligonal a costa de limitar algunas características, en este caso la posibilidad de aplicar iluminación.

En Bullet 3D tenemos 2 tipos de fuentes de luz distintas: La luz ambiental, que es única y afecta uniformemente a toda la escena y la luz puntual, que podemos tener varias y afectan dentro de un radio. Estas últimas son más sofisticadas y las veremos más adelante, por el momento vamos a ocuparnos de la luz ambiental, ya que es sencilla de usar y prácticamente obligatoria si queremos ver correctamente las texturas de nuestros modelos.

La instrucción `M8E_SETAMBIENTLIGHT ( )` recibe 4 parámetros que indican los valores de alpha, red, green y blue que queremos dar a la luz ambiental. El valor del alpha es ignorado ya que en la luz no tiene sentido una transparencia, mientras que el resto de parámetros toman valores entre 0 y 255 indicando la saturación de cada color.

Si queremos una luz normal podemos utilizar:

```
M8E_SETAMBIENTLIGHT ( 100 , 100 , 100 , 100 );
```

Esto produce una iluminación de aproximadamente el 40% del máximo, con la misma tonalidad en todos los componentes, es decir, luz blanca tenue. Observa que si nos excedemos con la iluminación ambiental las luces puntuales que añadamos más adelante se apreciarán menos, ya que no es posible superar el valor de 255 de iluminación en ninguna de las componentes de color, por eso se recomienda en general trabajar con colores de luz ambiental suaves.

## 2.4 Escalar el terreno

Si todo funciona correctamente ya deberías tener tu terreno en la escena y visualizar sobre el correctamente la textura, pero tenemos otro problema, el terreno es demasiado pequeño y las elevaciones del mapa resultan excesivas.

Esto se debe a que el terreno tiene una resolución en píxeles generalmente pequeña en comparación con el tamaño de otros elementos como el anterior mapa de Quake o el cubo. Es habitual tener que modificar los tamaños de algunos modelos 3D, y el mapa de terreno no es una excepción.

Para escalar un modelo, en cualquiera de los 3 ejes, podemos usar la instrucción `M8E_MODELSCALE ( )`, que recibe como primer parámetro un identificador válido de modelo cargado y como siguientes parámetros la escala en los ejes x, y, z, respectivamente, medida en tanto por 1. La escala es un número real (float), por lo que podemos introducir valores decimales como por ejemplo 0,5 para escalar al 50% del tamaño original.

Los mapas de terreno generalmente conviene escalarlos bastante en los ejes x, z para estirarlos, mientras que en el eje y no es necesaria mucha escala ya que las alturas del terreno suelen ser suficientemente significativas.

Un ejemplo de escalado del terreno sería:

```
M8E_MODELSCALE ( mapa , 8 , 4 , 8 );
```

## 2.5 Crear un módulo secundario

Todo lo que hemos hecho hasta ahora es totalmente correcto, pero a largo plazo es una mala costumbre comenzar a añadir instrucciones en el fichero de código principal videojuego.prg

Por eso vamos a crear un módulo de código secundario dentro del subdirectorio /prg en el cual vamos a añadir las instrucciones de carga del terreno, escalado del mismo e iluminación ambiental. Una forma sencilla de crear un nuevo fichero .prg es copiar uno de los que ya existen, renombrarlo y dejarlo en blanco. En este caso llamaremos a nuestro módulo mapa.prg

En primer lugar y para que no se nos olvide, vamos a añadir una nueva sentencia INCLUDE en videojuego.prg indicando que debe hacer uso del nuevo módulo, sería:

```
INCLUDE "prg/mapa.prg";
```

Por ahora el orden de las sentencias INCLUDE no tiene que preocuparnos demasiado, aunque siempre es bueno mantener cierto orden y dado que el módulo del mapa es un módulo de más alto nivel que los demás, convendría cargarlo en último lugar.

Una vez hecho esto ya podemos editar el fichero mapa.prg y crear en él un sencillo proceso que se encargue de realizar todas las instrucciones de carga de nuestro terreno. Recuerda que al tratarse de instrucciones deben ir dentro de un bloque BEGIN END, que es lo que formalmente se llama "cuerpo" del proceso.

Un esqueleto de proceso en el que podríamos añadir nuestras instrucciones sería:

```
PROCESS mapa ()  
BEGIN  
    // Aquí vendrían las instrucciones  
END
```

Una vez editado el módulo mapa.prg con las instrucciones de carga del terreno sólo nos falta un detalle: En Bennu los procesos deben ser invocados para ejecutarse, lo que hemos hecho ha sido solamente declarar el proceso, lo invocaremos desde el fichero de código principal videojuego.prg, por ejemplo en el mismo punto en el que antes cargábamos el mapa de Quake.

La invocación del proceso se realizaría así:

```
mapa ();
```