

## 43. Modelos 3D

Nuevamente, Benu 3D es compatible con todos los modelos 3D asociados a Quake III y otros juegos compatibles, pero también nos ofrece soporte para otros formatos de modelo 3D habituales. La lista completa de modelos 3D puede consultarse en la documentación de Benu 3D (Subdirectorio /doc), pero en este tutorial solamente trataremos algunas de las extensiones que mejor resultado y compatibilidad ofrecen.

### 43.1 Formatos de Modelo 3D

Los diferentes formatos de modelo 3D pueden o no incluir distintas animaciones del modelo en el mismo fichero y tener mejor o peor calidad. Los formatos con los que trabajaremos en este tutorial y las características principales de cada uno son las siguientes:

Formato	Animación	Calidad	Características
.md2	Sí	Media	Modelos sencillos de Quake II muy fáciles de encontrar en internet.
.md3	Sí	Alta	Modelos de Quake III, buena calidad pero no son fáciles de encontrar en internet.
.obj	No	Alta	Modelos de utilidad general, es un formato compatible con la mayoría de programas de diseño 3D.
.3ds	No	Alta	Modelos de 3D Studio, uno de los programas de diseño 3D más utilizados.
.x	No	Media	Modelos sencillos, no todos ofrecen compatibilidad con Benu 3D así que los probaremos antes de usarlos.

La disponibilidad de animaciones en el modelo 3D es esencial para personajes que vayan a tener cierto protagonismo o "vida propia" en nuestro videojuego. Para este fin utilizaremos principalmente modelos md2 dada la gran variedad de los mismos que encontraremos en internet.

En cambio, para modelos estáticos que puedan servir como decoración y no precisen de animación, podemos hacer uso del resto de formatos.

### 43.2 Cargar modelos 3D

La carga de modelos 3D "pequeños" (Recuerda que para modelos muy grandes y/o escenarios podemos usar las funciones del tema anterior), se realiza con una sencilla función:

```
int M8E_LOADANIMODEL ( string );
```

El parámetro que recibe es la ruta del fichero con el modelo 3D a cargar, que debe tener uno de los formatos soportados/recomendados.

La función nos retorna el identificador del modelo cargado, que deberemos almacenar dentro de una variable para luego aplicarle otras funciones para rotarlo, escalarlo, animarlo, etc.

Observa que la carga del modelo 3D no especifica nada sobre su posición, así que consideraremos que M8E\_LOADANIMODEL ( ) simplemente cargará en memoria los modelos a utilizar y será necesario que le apliquemos una serie de funciones para ubicarlo en el escenario.

A diferencia de Bennu 2D, en el que las instrucciones de tipo load\_xxx ( ) se ejecutaban al principio y una única vez por cada fichero a cargar, permitiendo que varios procesos distintos hagan uso del identificador de fichero cargado, en Bennu 3D tendremos que cargar en distintas variables el mismo modelo para tener varios procesos que hagan uso de él.

### 43.3 Aplicar modificaciones básicas al modelo 3D

Como ya comentamos en el tema 41, las operaciones básicas que podemos realizar en 3D sobre un modelo son las rotaciones, escalados y traslaciones, en cualquiera de los 3 ejes.

Para facilitar el trabajo con 3 coordenadas, Bennu 3D tiene predefinido un tipo de dato llamado \_3dpos que es un TYPE con los campos float x, y, z. Muchas de las funciones de Bennu 3D requieren que se les pase una variable de tipo \_3dpos con los valores 3D que queremos aplicar.

Todas las operaciones básicas se realizan sobre el identificador de modelo devuelto por M8EE\_LOADANIMODEL ( ), y las funciones que se encargan de ello son las siguientes:

```
M8E_MODELROTATION ( int, _pos3d );
```

Recibe como primer parámetro el identificador de modelo que queremos rotar, y como segundo parámetro una variable de tipo \_pos3d en cuyos campos x, y, z debemos haber asignado el valor de rotación que queremos para el correspondiente eje. En este caso, los valores de rotación deben ser especificados en grados (No en milésimas), pero al ser campos de tipo float podemos establecer los decimales que queramos para la rotación.

`M8E_MODELSCALE ( int, float, float, float );`

Recibe como primer parámetro el identificador de modelo que queremos escalar, y como siguientes parámetros los valores de escalado en los ejes x, y, z que queremos aplicar al modelo. Observa que en este caso no es necesario utilizar una variable de tipo `_pos3d`, sino 3 valores `float`.

`M8E_POSMODEL ( int , _pos3d );`

Recibe como primer parámetro el identificador de modelo que queremos escalar, y como segundo parámetro una variable de tipo `_pos3d` en cuyos campos x, y, z debemos haber asignado el valor de traslación que queremos para el correspondiente eje.

Para la aplicación de las funciones anteriores, observa que también los escenarios vistos en el tema 42 retornan un identificador de modelo, y estas funciones también son aplicables a ellos, así que podemos rotar, escalar y desplazar el mapa durante la ejecución.