(Mode8 Enhanced Engine M8ee for BennuGD)

Actualizado: 12/11/2010

http://coldev.cjb.net
http://coldev.blogspot.com

Ultima actualización: http://trinit.es
Traducción a español: http://trinit.es

TRINT
ASOCIACION DE INFORMATICOS DE ZARAGOZA

Índice

Introducción

Bennu3D permite usar los motores Irrlicht y Bullet 3D (Copyright (C) 2002-2004 Nikolaus Gebhardt) con Bennu.

Licencia: GNU/GPL

Formatos de textura actualmente soportados:

Adobe Photoshop (.psd) (No testeado)
JPEG File Interchange Format (.jpg) Compatible 100%!
Portable Network Graphics (.png) Compatible 100%!
Truevision Targa (.tga) Compatible 100%!
Windows Bitmap (.bmp) Compatible 100%!
Zsoft Paintbrush (.pcx) Compatible 100%!

* Formatos de modelos 3D actualmente soportados:

3D Studio meshes (.3ds) Buena compatibilidad! (No testeado a fondo)
Blitz3D B3D files (.b3d) (No testeado)
Alias Wavefront Maya (.obj) Buena compatibilidad! (No testeado a fondo)
Cartography shop 4 (.csm) (No testeado)
COLLADA (.xml, .dae) *Mala compatibilidad, no siempre se encuentran en formato de hoja de cálculo .xml*
DeleD (.dmf) (No testeado)
FSRad oct (.oct) (No testeado)
Irrlicht scenes (.irr) (No testeado)
Irrlicht static meshes (.irrmesh) (No testeado)
Microsoft DirectX (.x) (binary & text) Mala compatibilidad, errores de carga.
Milkshape (.ms3d) Buena compatibilidad! (No testeado a fondo)
My3DTools 3 (.my3D) (No testeado)
OGRE meshes (.mesh) (No testeado)
Pulsar LMTools (.lmts) (No testeado)
Quake 3 levels (.bsp) Compatible 100%!
Quake 3 models (.md3) Compatible 100%!
Quake 2 models (.md2) Compatible 100%!
Other Quake models (.mdl) Compatibilidad intermedia, a veces causa problemas con las texturas.
STL 3D files (.stl) (No testeado)

Glosario de funciones del motor Bullet 3D para BennuGD

Funciones de inicialización y cierre del motor

M8E_INIT (int driver)

Inicializa la librería 3D. Es la primera función que debe ser invocada para empezar a utilizar cualquier otra función del motor 3D. El parámetro driver debe ser una de las siguientes CONST predefinidas: EDT_NULL, EDT_SOFTWARE, EDT_BURNINGSVIDEO, EDT_DIRECT3D8, EDT_DIRECT3D9, EDT_OPENGL.

EDT_NULL vale 0, así que puedes usar 0 como parámetro para utilizar el driver por defecto, mientras que EDT_OPENGL vale 5, así que puedes usar un 5 como parámetro para utilizar OpenGL, que es el driver que más calidad ofrece ahora mismo, especialmente si trabajamos con aspectos gráficos avanzados como nieblas, emisores de partículas, iluminación, etc.

En caso de tener limitaciones exhaustivas dadas por al GPU es recomendable utilizar EDT_BURNINGSVIDEO, ya que obtiene una calidad aceptable utilizando únicamente CPU, aunque algunos aspectos avanzados como iluminación en tiempo real y niebla se verán deshabilitados.

int M8E_FREE ( )

Finaliza el uso de la librería 3D y libera todos los recursos. Es la última función que debes llamar, por ejemplo antes de cerrar el programa, para terminar de usar el motor 3D y liberar correctamente todos los recursos cargados con M8E_INIT ( ).

Int M8E_SCREENEXIST ( )

Devuelve TRUE si la librería se cargó correctamente, puede usarse después de M8E_INIT ( ) para comprobar que se cargó bien.

Int M8E_OPTIONEXIST (int)

Pone en la cola las características de la tarjeta de vídeo. Su funcionalidad no está bien definida en las primeras versiones de 2010.

int M8E_GETDRIVER ( )

Obtiene el actuar driver de video y devuelve una de las siguientes CONST: EDT_NULL, EDT_SOFTWARE, EDT_BURNINGSVIDEO, EDT_DIRECT3D8, EDT_DIRECT3D9, EDT_OPENGL cuyos valores enteros van de 0 a 5 al igual que en la función M8E_INIT ( ).

int M8E_GETFPS ( )

Devuelve el valor actual de frames por segundo a los que el programa está ejecutándose. Idealmente 60 frames por segundo garantiza una fluidez total, valores por debajo implican una cada vez mayor pérdida de calidad mientras que valores por encima resultan inapreciables y generalmente no hacen más que consumir recursos innecesarios.

Funciones matemáticas

M8E_ADD3DLINE(_Line3D line)

Crea una línea en 3D en el espacio. La estructura _Line3D tiene un campo de tipo _pos3D llamado pstart y otro campo, también de tipo _pos3D llamado pend que indican el punto de inicio y final de la linea, respectivamente. En las primeras versiones de Bennu 3D en 2010 no realiza el dibujado de la linea.

FLOAT M8E_GETDISTANCE (int model1 , int model2)

Devuelve la distancia entre dos identificadores de modelo válidos pasados como parámetro.

FLOAT M8E_GETDISTANCESQ(INT model1, int model2)

Devuelve la raíz cuadrada de la distancia entre dos identificadores de modelo válidos pasados como parámetro. Es útil para realizar cálculos trigonométricos sobre la distancia real entre modelos.

Funciones de iluminación y efectos ambientales

M8E_SETAMBIENTLIGHT (int alpha , int R , int G , int B)

Establece la luz ambiental para toda la escena. Los últimos parámetros indican el color RGB de la luz, el primer parámetro indica la transparencia, pero no tiene ninguna utilidad actualmente. Todos los valores de los parámetros deben estar comprendidos entre 0 y 255, valores fuera de ese intervalo son truncados y por tanto dan lugar a comportamientos extraños.

Nótese que es necesario que haya alguna luz en la escena para que los modelos cargados muestren su color real, en otro caso se verán totalmente negros. Es recomendable tener una luz ambiental no demasiado alta (p.ej. 100, 100, 100) para que las luces puntuales tengan un efecto notable.

La luz ambiental es única, por tanto sucesivas llamadas a esta función con distintos valores cambiarán el valor anterior de la luz ambiental por el último dado.

M8E_SETFOG (int r , int g , int b , float start , float end)

Establece niebla en la escena con un color RGB y un degradado respecto de un punto de inicio y fin, todos estos valores se indican como parámetros en ese mismo orden.

Sólo puede haber una niebla activa al mismo tiempo y ésta se generará tomando como referencia la cámara activa, sucesivas llamadas a esta función eliminarán la niebla anterior para generar la actual. Es posible realizar llamadas sucesivas a esta función con valores ligeramente distintos para modificar los valores de la niebla gradualmente e incluso simular efectos como amaneceres y atardeceres.

Nótese que los modelos cargados serán afectados por la niebla sólo si su flag de material EMF_FOG_ENABLE está activado. Ver M8E_MODELSETMATERIAL ( ) para más información.

INT M8E_ADDSKYBOX (int up , int down , int left , int right , int front , int back)

Carga una caja de cielo con texturas. Todos los parámetros deben ser identificadores válidos de texturas cargadas, en otro caso se mostrará el color de fondo por defecto del render.

La caja de cielo es un cubo situado siempre a una gran distancia de la cámara de forma que nunca podamos llegar a él. Se compone de 6 texturas cuadradas, una por cada lado. Para evitar el efecto de corte que crean las aristas de la caja es preferible utilizar la siguiente función M8E_ADDSKYDOME ( ).

Devuelve un identificador de modelo cargado que puede utilizarse como cualquier otro identificador de modelo. Es especialmente útil para aplicar rotaciones y generar efectos de nubes y cielo en movimiento.

INT M8E_ADDSKYDOME (int texture , int horizRes , int vertRes , float texturePercentage , float spherePercentage)

Carga una caja esférica de cielo con textura. La caja esférica actúa como una caja de cielo normal, pero tiene una única textura que se muestra en la esfera. La sensación es más realista que con una skybox, ya que todos los puntos de la esfera están a la misma distancia del centro, mientras que en el caso de la skybox las aristas están más alejadas y pueden verse más afectadas por la niebla.

HorizRes y VertRes indican el numero de lados de cada diámetro de la Skydome, a mayores valores mayor calidad, con 16 se alcanza una calidad mas que aceptable. TexturePercentage indica en tanto por 1 la cantidad de textura que ocupara la totalidad de la pantalla, con 1 veremos la totalidad de la Skydome. SpherePercentage debe tomar valores mayores estrictos que 1 para lograr una curvatura esférica, con el valor 2 se alcanza una visión perfecta del Skydome, mientras que con valores menores se puede obtener un efecto de cilindro.

Devuelve un identificador de modelo cargado que puede utilizarse como cualquier otro identificador de modelo. Es especialmente útil para aplicar rotaciones y generar efectos de nubes y cielo en movimiento.

INT M8E_ADDLIGHT (float posX, float y, float z, float red, float green, float blue, float radius)

Añade una luz en la escena, los 3 primeros parámetros indican la posición, los 3 siguientes indican el color RGB de la luz y el último parámetro indica el radio de iluminación. Retorna un identificador válido de la luz.

Nótese que los modelos cargados por defecto no tienen reacción a este tipo de iluminación, para ello es necesario aplicar el tipo de material apropiado con M8E_MODELMATTYPE ( ) sobre el modelo. En general los materiales a partir del 6 ofrecen reacción a la iluminación en tiempo real con diferente calidad y efectos asociados.

Devuelve un identificador de luz puntual que puede posicionarse o eliminarse de la misma manera que un modelo cargado.

M8E_LIGHTGETPROP (int light , _Plight POINTER prop)

Obtiene las propiedades de la luz dada como parámetro mediante una referencia a una estructura _Plight. Esta función no está totalmente testeada en las primeras versiones de 2010. Los campos de la estructura _Plight son los siguientes:

```
_PColor AmbientColor;     // Indica en los campos R, G, B el color de la luz ambiental.
_Pos3D Attenuation;       // Indica en los campos X, Y, Z la atenuación lumínica en cada eje.
int CastShadows;          // Indica con TRUE/FALSE si la luz proyecta sombras.
_PColor DiffuseColor;     // Indica en los campos R, G, B el color de luz difusa.
_Pos3D Direction;         // Indica en los campos X, Y, Z la dirección de emisión del foco de luz.
float Falloff;            //
float InnerCone,          // Indica el tamaño del foco interior del cono de luz.
float OuterCone;          // Indica el tamaño del foco exterior del cono de luz.
_Pos3D Position;          // Indica en los campos X, Y, Z la posición de la luz en el mundo.
float Radius;             // Indica el radio total de iluminación.
_Pcolor SpecularColor;    // Indica en los campos R, G, B el color de luz especular.
int LightType;            // Indica con las constantes ELT_POINT = 0, ELT_SPOT = 1, ELT_DIRECTIONAL = 2 si
                          // se trata de una luz puntual, de foco o direccional.
```

M8E_LIGHTSETPROP (int light , _Plight prop)

Establece las propiedades de la luz dada como parámetro mediante una estructura _Plight. Sus campos han sido definidos en la función anterior.

Funciones de carga y descarga de modelos y texturas

int M8E_LOADCUBEMODEL (float size)

Carga un modelo cúbico con un tamaño de arista dado como parámetro. Devuelve un identificador de modelo válido.

int M8E_LOADSPHEREMODEL (float radius)

Carga un modelo esférico con un radio dado como parámetro. Devuelve un identificador de modelo válido.

Int M8E_ADDZIPFILE (string filepath)

Añade un fichero comprimido en formato .zip al sistema de archivos, de esta forma todo el contenido del fichero podrá ser referenciado como si estuviese descomprimido en el directorio principal del programa. Devuelve TRUE si se cargó correctamente.

Se utiliza especialmente para cargar mapas de Quake 3, cuyo formato de fichero .pk3 es un simple renombrado del formato .zip. Estos mapas se comprimen para evitar ocupar excesivo espacio en disco.

int M8E_LOADTERRAIN (string heightmap)

Añade un terreno a partir de la ruta de un textura con un mapa de alturas en el que las zonas más claras presentan terreno elevado y las zonas más oscuras terreno menos elevado. Los mapas de alturas suelen encontrarse en escala de grises, y pueden ser generados fácilmente a partir de cualquier textura aplicando el efecto de blanco y negro y cualquier tipo de difuminado para suavizar el terreno, ya que cambios bruscos de color generarán alteraciones bruscas de la altura.

Devuelve un identificador válido de modelo que puede ser tratado como cualquier otro modelo.

Int M8E_LOADTILETERRAIN (int width , int height , float tileSize , int meshSize)

Añade un terreno tileado aleatorio, útil para terrenos grandes, pero requiere mas memoria. Los valores width y height indican las distancia en altura y en longitud de las variaciones aleatorias en el terreno generado, de forma que a mayores valores se obtiene mayor suavidad. tileSize indica el tamaño de cada tile y meshSize la cantidad de tiles que se generarán a lo ancho y a lo largo, que debe ser menor e igual que 8 para evitar desbordamientos de memoria.

Actualmente tiene una limitación en el nombre de la textura que se carga por defecto en el terreno generado, que debe encontrarse en el subdirectorio /media y tener como nombre dirt_x_43.bmp

int M8E_MODELCLONE (int model)

Clona un modelo válido previamente cargado que viene dado como parámetro. Retorna el identificador del nuevo modelo clonado a partir del modelo original.

Es especialmente útil a la hora de cargar varias veces el mismo modelo, ya que evita el acceso a disco y lo clona directamente a partir del modelo cargado en memoria, ganando eficiencia.

M8E_MODELREMOVE (int model)

Elimina un modelo cargado cuyo identificador es dado como parámetro. Si el modelo había cargado algún modelo adicional con M8E_MODELADDCHILD ( ) éste también será removido. Nótese que en caso de aplicar esta función sobre un modelo que no existe o que ya ha sido eliminado previamente se produce un error de ejecución.

Int M8E_LOADTEXTURE (string filepath)

Carga una textura cuya ruta de fichero es dada como parámetro, retorna un identificador de textura válido que puede ser usado en otras funciones como M8E_LOADTEXMODEL ( ) para cubrir un modelo con la textura, o con las funciones de GUI para añadir elementos 2D a la pantalla. Nótese que una misma textura cargada puede aplicarse sobre tantos modelos como se desee.

Los formatos de imagen son todos los soportados por Bennu 3D.

M8E_LOADTEXMODEL (int model, int textureID)

Establece una textura cargada cuyo identificador es dado como segundo parámetro sobre un modelo cargado cuyo identificador es dado como primer parámetro. El identificador de la textura puede ser único y ser usado en tantos modelos cargados como se desee.

M8E_MODELSETID (int model , int pid)

Establece sobre el identificador de modelo dado como parámetro una etiqueta identificativa de tipo entero. Es especialmente útil para añadir al modelo cargado el identificador del proceso que lo gestiona, así es sencillo obtener el identificador de proceso asociado a un modelo detectado mediante trayectoria de rayo con M8E_RAYCOL ( ).

int M8E_MODELGETID (int model)

Devuelve la etiqueta identificativa asociada al modelo dado como parámetro. Esta etiqueta ha debido ser previamente establecida usando M8E_MODELSETID ( ).

M8E_LOADTEXMODELA (int model, int index , int textureID)

Establece una textura de material en un modelo multitextura. Se usa con shaders.

Int M8E_LOADMATRIX (float tileHeight , float tileWidth , int tileCountHeight , int tileCountWidth , float HillsHeight , float countHillsheight , float countHillsWidth , float textureRepeatCountHeight , float textureRepeatCountWidth)

Añade un modelo de plano de colinas que se utiliza entre otras cosas para posicionar agua sobre él utilizando la función M8E_LOADWATER ( ). Devuelve un identificador de modelo. Los 2 primeros parámetros indican la anchura y profundidad de cada tile del plano. Los 2 siguientes valores indican el numero de tiles en anchura y profundidad. Conviene usar tiles suficientemente grandes y menos cantidad de los mismos para mejorar la eficiencia. Los 3 siguientes valores indican la altura de las colinas (Si las hay, 0 para agua) y la cantidad de las mismas en anchura y profundidad. Los 2 últimos valores indican la cantidad de veces que se repetirá la textura en anchura y profundidad.

int M8E_LOADWATER (int matrixmodel , float waveHeight , float waveSpeed , float waveLength)

Añade un modelo para renderizar una superficie de agua animada. Requiere una matriz de modelos (M8E_LOADMATRIX) en el primer parámetro sobre la cual se posicionara el modelo de agua. Devuelve un identificador de modelo. Los siguientes parámetros indican la altura de las olas, el tiempo entre olas y la longitud de las olas.

int M8E_LOADANIMODEL (string pathfile)

Carga un modelo bien sea estático o animado, el fichero debe tener uno de los formatos soportados por Bennu 3D y su ruta de fichero completa y con extensión debe venir dada en el parámetro. Es la función principal para la carga de modelos pequeños (No sirve para mapas). Devuelve un identificador válido de modelo.

int M8E_LOADMODELEX (string pathfile , int detailLevel)

Carga modelos grandes y modelos estáticos en general, el fichero debe tener cualquiera de los formatos soportados. Perfecto para edificios y mapas. El nivel de detalle puede ir desde 0 hasta 255, siendo 255 el mayor nivel de detalle, aunque no se aprecian diferencias significativas en la calidad ni en el rendimiento y conviene cargar siempre con calidad máxima. Devuelve un identificador de modelo.

int M8E_LOADQ3MAP (string pathfile , int collisionList)

Carga un mapa de Quake 3 Arena con shaders internos incluidos, también puede añadirse una lista de colisiones como parámetro (Opcional). Puede usarse un valor 0 en la lista de colisiones para cargarlo sin soporte para colisiones.

M8E_MODELADDCHILD (int model , int modelChild)

Establece un modelo "hijo" en otro modelo. Esto significa que el modelo "hijo" siempre seguirá la posición, rotación y escala del modelo principal. Es especialmente útil para añadir ropas o armas a un modelo, especialmente en el caso de modelos .md2.

Nótese que los estados de animación (Si existen) deben ser controlados para ambos modelos, el principal y el "hijo". Por ejemplo, si el modelo principal está en el estado de animación EMAT_RUN, el modelo "hijo" debe cambiar su estado de animación a EMAT_RUN al mismo tiempo.

M8E_CLEARSCENE ( )

Elimina todos los objetos y limpia la pantalla, puede usarse para finalizar el programa e incluso para hacer un sencillo cambio de fase. Se recomienda descargar todos los ficheros cargados y matar a todos los procesos activos antes de realizar esta llamada.

Funciones de posición, rotación y escalado de modelos

M8E_POSMODEL (int model , _Pos3D pos)

Establece la posición de un modelo en el escenario a partir de las coordenadas 3D dadas como parámetro. La variable de tipo Pos3D tiene 3 campos de tipo float llamados x, y, z que representan cada una de las coordenadas. La posición será asignada de forma relativa al origen de coordenadas, que es la forma más habitual. Si se desea asignar la posición de manera absoluta respecto del propio modelo hay que usar M8E_POSMODELXYZ ( ).

M8E_POSMODELXYZ (int model , _Pos3D pos)

Establece la variación de posición de un modelo en el escenario a partir de las coordenadas 3D dadas como parámetro. La variación de posición dada como parámetro no será asignada directamente, sino que será aplicada como incremento en el frame actual, siempre respecto de los ejes del modelo, por lo que resulta especialmente útil para conseguir avances en los ejes del modelo cuando se han aplicado sobre él rotaciones complejas.

M8E_GETPOSMODEL (int model , _Pos3D POINTER pos)

Obtiene la posición real de un modelo en el escenario en la referencia a las coordenadas 3D dada como parámetro. Resulta útil cuando se aplican colisiones, ya que es posible que los valores numéricos de las variables de posición no coincidan con los valores reales cuando hay bloqueo por colisión. En esos casos conviene utilizar esta función después de FRAME para conservar el valor real de la posición en todo momento.

M8E_MODELROTATION (int model , _Pos3D rot)

Establece la rotación de un modelo en el escenario a partir de las coordenadas 3D dadas como parámetro, en grados. La rotación se realiza respecto del origen de coordenadas en la magnitud que se indique, es poco recomendable para rotaciones en varios ejes, ya que requiere aplicar trigonometría para conseguir rotaciones según los ejes del modelo. Advertencia: En las primeras versiones de 2010 hay un bug y la rotación en X se aplica sobre el eje del modelo, mientras que las rotaciones en Y, Z se aplican según los ejes del mundo. En conclusión, para rotaciones sobre varios ejes es más aconsejable utilizar M8E_MODELROTATIONXYZ ( ).

M8E_MODELROTATIONXYZ (int model , _Pos3D rot)

Establece la variación de rotación de un modelo respecto de sus ejes a partir de las coordenadas 3D dadas como parámetro, en grados. Nótese que los valores de rotación no son asignados directamente, sino que son aplicados como incrementos, esto quiere decir que para mantener el modelo sin rotar debería asignarse 0 en todos los campos, mientras que los valores distintos de 0 provocarían rotación en los ejes correspondientes para el frame actual.

M8E_GETROTMODEL (int model , _Pos3D POINTER pos)

Obtiene la rotación real de un modelo en el escenario en la referencia a las coordenadas 3D dada como parámetro, en grados.

M8E_MODELSCALE (int model, float x, float y, float z)

Escala el modelo en los ejes 3D. Los valores x, y, z indican al modelo cuántas veces debe ser más grande (Si el valor es mayor que 1) o más pequeño (Si el valor es menor que 1), en cada eje. Valores negativos provocarán que el modelo invierta las normales de sus polígonos, sea espejado en todos sus ejes y aplique la escala en valor absoluto.

Animadores de movimiento automático

M8E_ADDANIMCIRCLE (int model ,float x , float y , float z , float radius , float speed)

Crea un animador circular. El modelo cargado cuyo identificador es dado como primer parámetro rotará alrededor de un punto cuya posición 3D es dada por los parámetros x, y, z. El siguiente parámetro indica el radio de rotación y el último parámetro indica la velocidad de giro. Nótese que la velocidad de giro es un indicador del número de rotaciones por segundo, por lo que es recomendado usar valores reales entre 0 y 1 para evitar rotaciones extremadamente rápidas.

M8E_ADDANIMLINE (int model , float x , float y , float z , float x2 , float y2 , float z2 , int TimeMs , int loop)

Crea un animador de vuelo en linea recta, que permite al modelo asociado volar o desplazarse a través de una linea entre 2 puntos. El primer parámetro indica el identificador del modelo cargado que se moverá por la línea, los siguientes 3 parámetros son la posición inicial y los 3 siguientes la posición final, el siguiente parámetro indica el tiempo (En milisegundos) que tomará para volar sobre la linea, y finalmente el último de los parámetros indica con TRUE o FALSE si la animación debe repetirse continuamente o no.

Nótese que el movimiento será totalmente lineal y uniforme, y que la repetición del movimiento no provocará un movimiento a la inversa, sino que repetirá la misma linea trayendo el modelo desde la posición final hasta la inicial una vez finalizado el recorrido, por lo que el efecto puede resultar bastante brusco si se mantiene visible.

M8E_ADDANIMROTATION (int model , float x , float y , float z)

Crea un animador de rotación que rotará el modelo asociado sobre sí mismo. El primer parámetro indica el identificador del modelo cargado, y los 3 siguientes indican la velocidad de rotación en cada eje, de manera que el modelo puede rotar en varios ejes al mismo tiempo con diferente velocidad de rotación en cada uno.

M8E_ADDANIMDELETE (int model , int timems)

Borra el animador asociado a un modelo transcurrido un tiempo en milisegundos automáticamente.

int M8E_ADDPOINTLIST ( )

Crea una lista de puntos 3D y retorna un identificador válido de la misma. El identificador de la lista puede ser usado en las funciones M8E_POINTLISTADD ( ) y M8E_ADDANIMPOINTS ( ) para hacer que un modelo se mueva a través de sus putnos recorriendo una trayectoria en spline (Un spline es una curva suave entre varios puntos).

M8E_POINTLISTADD (int points3darray , float x , float y , float z)

Añade un punto 3D a un vector de puntos 3D creado con M8E_ADDPOINTLIST.

M8E_ADDANIMPOINTS (int model , int pointsArray , float speed , float tightness)

Asocia un vector de puntos 3D a un identificador de modelo dado como parámetro, los puntos en el vector 3D deben ser al menos dos y deben indicar los puntos a través de los cuales el modelo se moverá recorriendo un spline. Los últimos parámetros indican la velocidad de desplazamiento y suavidad con la que se recorrerá el spline.

El parámetro velocidad indica el tiempo en segundos que tomará el modelo para desplazarse de un punto a otro del spline, por esa razón es necesario tener en cuenta la distancia que se asigna entre puntos sucesivos, ya que puntos muy cercanos entre sí darán lugar a recorridos más lentos mientras que puntos muy alejados darán lugar a recorridos más rápidos.

Nótese que en un spline matemático, el valor de suavidad 0 indica un desplazamiento totalmente lineal entre los puntos, mientras que valores mayores que 0 realizarán curvas lo más suaves posibles entre ellos. También hay que tener en cuenta que valores demasiado elevados en el parámetro de suavidad producirán curvas demasiado pronunciadas que pueden desplazarse muy lejos de la nube de puntos, por lo que hay que aplicar este parámetro con cuidado.

Int M8E_ADDTEXTURELIST ( )

Crea una lista de texturas y devuelve un identificador válido de la lista que puede ser usado en las funciones M8E_TEXTURELISTADD ( ) y M8E_ADDANIMTEXTURES ( ) para aplicar texturas animadas a un modelo.

M8E_TEXTURELISTADD (int textureArray , int Texture)

Añade un identificador de textura cargada en una lista de texturas creada con M8E_ADDTEXTURELIST ( ), esta función se usa para crear un vector de texturas que puede ser usado para tener texturas animadas en un modelo usando M8E_ADDANIMTEXTURES ( ).

M8E_ADDANIMTEXTURES (int model , int texturesArray , int TimePerFrame , int loop)

Crea un animador de texturas, que intercambiará las texturas del modelo dado como parámetro basándose en las texturas del vector creado con M8E_ADDTEXTURELIST y M8E_TEXTURELISTADD. Puede especificarse el tiempo entre frames y si el vector de texturas debe ser cíclico mediante un TRUE/FALSE como último parámetro.

Una de las utilidades de esta función es la creación de zonas con agua donde la textura animada hace el efecto de oleaje.

M8E_MODELREMOVEANIM (int model)

Elimina todas las animaciones del modelo dado como parámetro. Esta función también elimina los manejadores de colisión.

Funciones de detección de colisiones

M8E_ADDCOLTERRAIN (int terrainModel)

Aplica una geometría real de colisiones sobre el identificador de modelo de terreno dado como parámetro. Esto servirá para añadirlo posteriormente a una lista de colisiones. Para manejar colisiones rápidamente es más aconsejable utilizar M8E_ADDCOLTERRAINFAST ( ).

M8E_ADDCOLTERRAINFAST (int model , int terrainmodel , float modelHeight , float NavMaxHeight , int ModelGlueEnable , float gravityX , float Y , float Z)

Crea un manejador rápido de colisiones con el terreno para un modelo. Debe especificarse en primer lugar el modelo y el identificador de mapa de terreno cargado para el cual se desea tener un manejador de colisiones. A continuación se especifica la altura que se desea establecer al modelo, que debe ser la suficiente para impedir que su parte más baja quede enterrada. El siguiente parámetro indica la máxima altura de navegación, que determina el valor máximo de posición en el eje Y que será controlado por este manejador, El siguiente valor indica si está activado el "modo pegamento", aquí indicamos con TRUE/FALSE si será posible controlar la posición en el eje Y por encima del terreno o bien siempre tendremos el modelo pegado a ras de suelo. Los siguientes 3 valores indican un valor de gravedad en los 3 ejes, aunque se recomienda aplicar gravedad de forma independiente.

Este manejador de colisiones controlará la posición en el eje Y de manera automática impidiendo que el modelo pueda quedar por debajo del terreno, aunque el valor asignado a su posición en ese eje resulte menor. En esos casos resulta recomendable actualizar manualmente el valor real de la posición en el eje y utilizando M8E_TERRAINGETHEIGHT ( ), ya que en caso contrario puede almacenar valores menores que la posición real.

int M8E_RAYCOL (_Line3D Line)

Encuentra el identificador de modelo dentro de una trayectoria dada por una linea 3D. La linea 3D es una estructura que consta de 2 campos pstart y pend que a su vez tienen los campos x, y, z, por lo que la linea viene dada por 2 puntos, el de inicio y el de fin. Para que la colisión con un modelo pueda ser detectada es necesario que al modelo le haya sido asignado un objeto de colisiones, por ejemplo utilizando M8E_ADDCOLBBOX ( ) u otra función similar.

Es especialmente útil para la detección de colisiones en las trayectorias de un disparo, ya que una velocidad de avance muy elevada hace improbable que se produzca una colisión física, pero sí que es probable que la colisión se de dentro de la linea de avance.

int M8E_ADDCOLLIST ( )

Crea una lista de colisiones vacía, perfecta para multicolisión entre varios modelos, devuelve un identificador válido de lista de colisiones en la que podemos añadir los modelos que deban responder físicamente entre sí, incluido el mapa.

M8E_COLLISTADD (int collisionList , int model)

Añade un modelo con colisión a la lista de colisiones. Es necesario que previamente se haya definido el tipo de geometría de colisión para ese modelo con una de las siguientes funciones.

Int M8E_ADDCOLMAP (int model)

Crea una objeto de colisiones en octree para el modelo dado como parámetro. Retorna un identificador válido de colisión. Sólo modelos cargados con M8E_LOADMODELEX, rápido con modelos grandes.

M8E_ADDCOLBBOX (int model)

Crea un objeto de colisión con forma de caja, perfecto para modelos pequeños y/o animados. Es necesario aplicar esta función sobre un modelo antes de poder añadirlo a la lista de colisiones si queremos que responda a ellas con geometría de caja y también si deseamos detectar cualquier tipo de colisión dentro de una trayectoria con M8E_RAYCOL ( ) o a partir de una posición 2D con M8E_GETMODELFROM2DPOS ( ).

M8E_ADDANIMCOL (int ColisionId , int model , float ellipsoidRadiusx1 , float y1 , float z1 , float gravityPerSecondx2 , float y2 , float z2 , float ellipsoidTranslationx3 , float y3 , float z3)

Crea un animador especial para hacer detección y respuesta de colisiones automática. En primer lugar debemos especificar la lista de colisiones, en segundo lugar el modelo en cuestión, los 3 siguientes parámetros son los radios del elipsoide geométrico que responderá a las colisiones, los 3 siguientes parámetros pueden indicar una gravedad constante que se aplicará sobre el modelo y finalmente los 3 últimos parámetros permiten trasladar la geometría de la elipsoide si se desea. Nótese que esta función aplica la geometría de colisiones, la gravedad y añade el modelo a la lista, todo en una única llamada.

M8E_ADDSCNCOL (int collisionList)

Añade todos los modelos cargados a la lista de colisiones dada como parámetro. No funciona correctamente en las primeras versiones de 2010.

Funciones de cambio de coordenadas

int M8E_GETMODELFROMCAM ( )

Devuelve un identificador de modelo que se encuentra dentro del campo de visión de la cámara activa.

int M8E_GETMODELFROM2DPOS (int x , y)

Devuelve un identificador de modelo a partir de una posición 2D de la pantalla. Para que sea posible obtener el identificador del modelo es necesario que a éste le haya sido aplicada previamente una colisión con M8E_ADDCOLBBOX.

Esta función resulta especialmente útil para obtener identificadores de modelo a partir de elementos 2D de la interfaz, especialmente en el caso del mouse nos permitirá seleccionar modelos con él.

M8E_GETPOS2DFROM3DPOS (_Pos3D Pos1 , _Pos2D POINTER Pos2)

Devuelve la posición 2D de la pantalla que corresponde a una posición 3D. Esta función es especialmente útil para colocar elementos 2D de la interfaz sobre elementos 3D del escenario, como por ejemplo nombres o cursores.

Nótese que es necesario que haya una cámara activa y que se haya realizado al menos un render antes de poder realizar este cálculo.

float M8E_TERRAINGETHEIGHT (int terrain , float X , float Z)

Devuelve la posición Y que corresponde a un terreno cargado con M8E_LOADTERRAIN ( ) en una determinada posición X, Z dada como parámetro. Es útil para posicionar con precisión modelos estáticos sobre el terreno sin necesidad de aplicar un manejador de colisiones y también para mantener actualizada la variable de posición Y cuando ésta es gestionada por un manejador de colisiones.

Funciones de render y dibujado

M8E_RENDER ( )

Hace el render de toda la escena, debe ser usado antes de FRAME en el bucle de control principal para ver el resultado final del render de la escena. El valor de alpha es ignorado, pero los valores red, green, blue indican el color de fondo de la escena.

Nótese que sólo un proceso debe estar realizando la operación de render a cada FRAME, ya que es la operación que más recursos consume. También es posible saltarse la operación de render si el FPS decae por debajo de ciertos valores.

En las primeras versiones del motor, esta función requería 4 parámetros indicando el valor de Alpha y RGB del fondo del render. Actualmente se mantiene siempre en blanco.

M8E_RENDERBEGIN ( )

Prepara la escena para el render. Es usado para hacer el render paso a paso, usando M8E_DRAW ( ) cada vez que quieres hacer render de un puerto de visión en modos de pantalla partida. Ver las funciones siguientes.

Como en el caso anterior, en las primeras versiones del motor, esta función requería 4 parámetros indicando el valor de Alpha y RGB del fondo del render. Actualmente se mantiene siempre en blanco.

M8E_SETVIEWPORT (int x0 , int y0 , int x1 , int y1)

Establece un nuevo puerto de vista, toda operación de render es realizada en este nuevo área que cubre la pantalla desde las coordenadas x0, y0 hasta las coordenadas x1, y1 dadas como parámetros.

Es útil si quieres dividir la pantalla en sectores y ver diferentes puntos de vista en cada uno. En ese caso primero debes establecer el puerto de vista, después llamar a M8E_RENDERBEGIN ( ), a continuación establecer la cámara activa para ese puerto de vista y finalmente llamar a M8E_DRAW ( ) para mostrar el resultado. Debes repetir la operación tantas veces como puertos de vista quieras tener en el render. Para terminar debes llamar a M8E_RENDEREND ( ) para mostrar el resultado final.

M8E_DRAW ( )

Dibuja una escena rápidamente, se usa con múltiples puertos de vista, para dibujar cada uno de ellos por separado. Se usa junto con M8E_RENDERBEGIN ( ).

M8E_DRAWGUI ( )

Dibuja los elementos de GUI (Imágenes en 2D, texto, etc.), se usa con múltiples puertos de vista que son dibujados cada uno de ellos por separado. Previamente es necesario determinar con M8E_SETVIEWPORT ( ) el puerto de visión donde mostrar la GUI, que debe ser la pantalla completa si se quiere mostrar la GUI de la misma forma que en pantalla completa..

M8E_RENDEREND ( )

Hace el render de toda la escena. Se usa junto con M8E_RENDERBEGIN ( ) para finalizar la operación de render.

M8E_SCRSHOT (string filepath)

Crea una captura de pantalla del último frame renderizado y lo guarda en el fichero cuyo ruta es pasada como parámetro. Si el fichero no existe la función se encarga de crearlo y si ya existía lo sobreescribe. También generará la ruta de carpetas si se especifica.

El formato del fichero gráfico será el especificado en la extensión. Actualmente ha sido testeada con los formatos de imagen soportados .bmp, .png, .jpg, .tga. El formato .gif no está soportado.

Funciones de animación de modelos md2

M8E_ANIMODEL (int model , int beginframe , int endframe)

Anima un modelo .md2 cuyo identificador es dado como parámetro. Establece los números de frame entre los cuales se realiza la animación de forma cíclica. Para realizar una animación determinada de modelos .md2 es más sencillo usar M8E_MODELANIMMD2 ( ). Nótese que esta función no es aplicable a otro tipo de modelos ya que puede producir errores de ejecución y pérdida de fps.

M8E_MODELANIMSPEED (int model , float speed)

Configura la velocidad de animación del modelo .md2 dado como parámetro estableciendo la velocidad con la cual la animación se reproduce. Nótese que la animación de modelos .md2 no está basada en FRAMES, sino en milisegundos, por lo que es independiente del FPS.

M8E_MODELANIMSETFRAME (int model , float frame)

Establece el frame de la animación actual para el modelo .md2 dado como parámetro.

M8E_MODELANIMMD2 (int model , int Md2animationID)

Establece una animación MD2. Sólo para modelos de Quake 2 (.md2) que tengan animaciones predefinidas (No todos, generalmente personajes y armas).

El primer parámetro debe ser un identificador válido de modelo cargado, mientras que el segundo parámetro debe de ser uno de los siguientes valores constantes:

```
EMAT_STAND = 0;                 // Quieto
EMAT_RUN = 1;                   // Correr
EMAT_ATTACK = 2;                // Atacar
EMAT_PAIN_A = 3;                // Recibir daño A
EMAT_PAIN_B = 4;                // Recibir daño B
EMAT_PAIN_C = 5;                // Recibir daño C
EMAT_JUMP = 6;                  // Saltar
EMAT_FLIP = 7;                  // Acrobacia
EMAT_SALUTE = 8;                // Saludar
EMAT_FALLBACK = 9;              // Caer de espaldas
EMAT_WAVE = 10;                 // Flirtear
EMAT_POINT = 11;                // Vacilar
EMAT_CROUCH_STAND = 12;         // Agachado
EMAT_CROUCH_WALK  = 13;         // Andar agachado
EMAT_CROUCH_ATTACK = 14;        // Atacar agachado
EMAT_CROUCH_PAIN = 15;          // Recibir daño agachado
EMAT_CROUCH_DEATH = 16;         // Morir agachado
EMAT_DEATH_FALLBACK = 17;       // Morir y caer de espaldas
EMAT_DEATH_FALLFORWARD = 18;    // Morir y caer de frente
EMAT_DEATH_FALLBACKSLOW = 19;   // Morir y caer de espaldas lentamente
EMAT_BOOM = 20;                 // Explotar
```

Nótese que esta función debe ser llamada únicamente cuando se desea cambiar el estado de la animación. Si se invoca repetidamente, la animación será reseteada al primer frame una y otra vez, así que debe controlarse cuando hay que cambiar el estado de animación o no.

Por ejemplo: M8E_MODELANIMMD2 (model , EMAT_RUN);

FLOAT M8E_GETMODELANIMCURFRAME (int model)

Obtiene el actual frame de animación del modelo dado como parámetro. Es útil cuando quieres cambiar un modelo "hijo" mientras el modelo principal está usando una animación, ya que puede obtener el frame actual del modelo principal y después aplicarlo al nuevo modelo "hijo" sin necesidad de resetear el estado de animación del modelo principal.

Funciones de control de las cámaras

Int M8E_GETACTIVECAM ( )

Devuelve el actual identificador de la cámara que se encuentra activa.

M8E_SETACTIVECAM (int camera)

Establece como cámara activa el identificador de cámara dada como parámetro.

M8E_GETCAMTARGET (int camera , _Pos3D POINTER pos)

Obtiene la posición 3D del objetivo de la cámara cuyo identificador es dado como primer parámetro, la posición se almacena en la referencia a la posición 3D dada como segundo parámetro.

int M8E_ADDCAM (float PositionX , float y , float z , float TargetX , float y , float z)

Añade una cámara estática con la posición y objetivo dados como parámetros, devuelve el identificador de la cámara creada. Nótese que las funciones de modificación de posición de este tipo de cámaras modificarán su posición en la escena, pero no el objetivo al que apuntan, para modificar el objetivo debe usarse M8E_CAMTARGET ( ).

int M8E_ADDCAMFOLLOW (int model , float height , float leash , float speed)

Añade una cámara de seguimiento (Cámara en 3ª persona) que seguirá en todo momento la posición del identificador de modelo dado como primer parámetro. Los últimos parámetros indican las opciones de seguimiento como altura sobre el modelo, distancia y velocidad de reacción. Informalmente esta cámara se denomina "cámara tipo Mario 64".

Esta función devuelve un identificador válido de cámara que puede ser usado para hacer referencia a esta cámara y obtener un identificador de modelo a partir de ella o establecerla como activa/inactiva.

Nótese que si se usa esta función no es necesario asignar ningún parámetro de desplazamiento o movimiento de cámara, esto quiere decir que esta función es invocada y funciona directamente sin necesidad de realizar ninguna llamada adicional.

int M8E_ADDCAMFOLLOW_A (int model , float relativePosX , float y , float z , float speed)

Añade una cámara de seguimiento (Cámara en 3ª persona) a un modelo que siempre seguirá el identificador del modelo dado como primer parámetro. Los últimos parámetros indican opciones de seguimiento como posición relativa x, y, z de la cámara respecto del modelo, y finalmente el último parámetro indica la velocidad de reacción.

Para un seguimiento desde atrás pueden servir los valores X, Y, Z: -200 , 100 , 0

Para un seguimiento lateral pueden servir los valores X, Y, Z: 0 , 0 , 200

Esta función devuelve un identificador de cámara válido que puede ser usado para hacer referencia a esta cámara y obtener un identificador de modelo a partir de ella o establecerla como activa/inactiva.

Nótese de nuevo que si usamos esta función no es necesario establecer ningún parámetro de movimiento o desplazamiento, esto quiere decir que cuando la función es llamada funciona directamente sin necesidad de realizar llamadas adicionales.

int M8E_ADDCAMFPS (_CameraFPS data)

Añade una cámara FPS (Primera persona o shooter) cuyo objetivo es controlado con el mouse. El parámetro es una estructura de tipo _CameraFPS y en esta llamada sí que debemos establecer valores en los campos de la estructura para indicar reacciones a teclas u otras opciones de movimiento.

Si esta cámara es añadida a una lista de colisiones junto con un escenario de Quake 3 garantiza una respuesta perfecta a las colisiones con el escenario y a la gravedad. Para ello se recomienda usar M8E_ADDANIMCOL ( ) sobre el identificador de la cámara.

Los campos que deben ser asignados a la estructura son los siguientes:

```
int turnspeed;          // Indica la máxima velocidad de giro (Límite) de la cámara.
int movespeed;          // Indica la velocidad de desplazamiento de la cámara.
int jumpspeed;          // Indica la velocidad de salto de la cámara.

// Los siguientes campos son normalmente asignados a key ( ) dentro de un LOOP e indican los controles
Int straferight;        // Indica con TRUE o FALSE si la cámara debe desplazarse hacia la derecha
int strafeleft;         // Indica con TRUE o FALSE si la cámara debe desplazarse hacia la izquierda
Int moveforward;        // Indica con TRUE o FALSE si la cámara debe moverse hacia adelante
Int moveback;           // Indica con TRUE o FALSE si la cámara debe moverse hacia atrás
Int jump;               // Indica con TRUE o FALSE si la cámara debe saltar
```

M8E_CAMTARGET (int camera , _Pos3D pos)

Establece el objetivo de la cámara en la posición 3D dada como parámetro.

M8E_CAMSETFAR (int camera , float value)

Establece la lejanía del plano horizonte de la cámara. Permite aumentar o reducir la distancia de visión, por defecto este valor está establecido a 2000 en todas las cámaras. Nótese que aumentar la distancia de visión disminuye el rendimiento.

M8E_CAMSETASPRATIO (int camera , float value)

Configura el radio de aspecto de la cámara (Proporción horizontal/vertical).

M8E_CAMSETFOV (int camera , float value)

Establece el campo de visión de la cámara (Ángulo de visión).

M8E_CAMSETNEAR (INT camera , float value)

Establece la distancia del plano más cercano de la cámara.

M8E_CAMSETUPVECTOR (int , float , float , float)

Configura totalmente el vector de aspecto de la cámara (Radio de aspecto, campo de visión y distancia de plano cercana).

Funciones de materiales y efectos de render

M8E_MODELSETMATERIAL(INT model, int materialFlag, int enable)

Establece un flag de material como activo o inactivo para un modelo cuyo identificador es dado como parámetro. El flag de material debe ser uno de los siguientes valores constantes:

```
EMF_WIREFRAME=1h;              // Muestra solamente aristas
EMF_POINTCLOUD = 2h;           // Muestra solamente vértices
EMF_GOURAUD_SHADING =4h;       // Muestra texturas y superficies con interacción lumínica
EMF_LIGHTING=8h;               // Muestra las texturas y superficies siempre iluminadas
EMF_ZBUFFER =10h;              // Muestra todas las superficies ocultas, incluso detrás de otros
                                  modelos
EMF_ZWRITE_ENABLE =20h;        // Toma en cuenta la profundidad para no dibujar si está oculto
EMF_BACK_FACE_CULLING=40h;     // Oculta las superficies traseras
EMF_FRONT_FACE_CULLING= 80h;   // Oculta las superficies frontales
EMF_BILINEAR_FILTER =100h;     // Usa filtro bilineal para suavizar texturas escaladas
EMF_TRILINEAR_FILTER =200h;    // Usa filtro trilineal para suavizar texturas escaladas
EMF_ANISOTROPIC_FILTER =400h;  // Usa filtro anti isotrópico para suavizar texturas escaladas
EMF_FOG_ENABLE=800h;           // Hace que el modelo sea afectado por la niebla
EMF_NORMALIZE_NORMALS= 1000h;  // Establece todos los vectores normales a valores unitarios
EMF_TEXTURE_WRAP = 2000h;      // ¿?
EMF_ANTI_ALIASING= 4000h;      // Usa anti aliasing para suavizar los bordes
EMF_COLOR_MASK= 8000h;         // No se ve nada si se usa este flag ¿?
EMF_COLOR_MATERIAL= 10000h;    // ¿?
```

El parámetro enable actuá como un booleano que indica si el flag actual debe ser activado o desactivado, al ser los valores numéricos potencias de dos es posible acumular varios efectos sobre el mismo modelo.

Por ejemplo: M8E_MODELSETMATERIAL(model, EMF_LIGHTING,false);

M8E_MODELVISIBLE (int model, int setvisible)

Establece un modelo dado como primer parámetro como visible o invisible, el segundo parámetro actuá como un booleano que indica visible (TRUE) o invisible (FALSE). Esto no implica que el modelo sea eliminado de la escena, simplemente dejará de ser interpretado por el render.

M8E_MODELMATTYPE (int model , int materialType)

Establece el tipo de material para las texturas en el identificador de modelo dado como primer parámetro. El tipo de material debe ser dado como segundo parámetro a través de uno de los siguientes valores constantes:

```
EMT_SOLID =0;
EMT_SOLID_2_LAYER =1;
EMT_LIGHTMAP =2;                          // Iluminado al 100%
EMT_LIGHTMAP_ADD =3;                      // Iluminado al 100%
EMT_LIGHTMAP_M2 =4;                       // Iluminado al 100%
EMT_LIGHTMAP_M4 =5;                       // Iluminado al 100%
EMT_LIGHTMAP_LIGHTING =6;
EMT_LIGHTMAP_LIGHTING_M2 =7;
EMT_LIGHTMAP_LIGHTING_M4 =8;
EMT_DETAIL_MAP=9;
EMT_SPHERE_MAP = 10;
EMT_REFLECTION_2_LAYER =11;
EMT_TRANSPARENT_ADD_COLOR = 12;           // Establece una transparencia aditiva del 50%
EMT_TRANSPARENT_ALPHA_CHANNEL =13;        // Interpreta el canal alpha como transparencia
EMT_TRANSPARENT_ALPHA_CHANNEL_REF =14;
EMT_TRANSPARENT_VERTEX_ALPHA =15;         // Invierte normales
EMT_TRANSPARENT_REFLECTION_2_LAYER =16;   // Establece una transperencia aditiva del 50%
EMT_NORMAL_MAP_SOLID =17;                 // Iluminado al 0%
EMT_NORMAL_MAP_TRANSPARENT_ADD_COLOR =18; // Transparencia 100%
EMT_NORMAL_MAP_TRANSPARENT_VERTEX_ALPHA =19; // Iluminado al 0%
EMT_PARALLAX_MAP_SOLID =20;               // Iluminado al 0%
EMT_PARALLAX_MAP_TRANSPARENT_ADD_COLOR =21; // Transparencia 100%
EMT_PARALLAX_MAP_TRANSPARENT_VERTEX_ALPHA =22;
EMT_ONETEXTURE_BLEND = 23;
EMT_FORCE_32BIT =24;
```

Nótese que sólo un tipo de material puede estar activo al mismo tiempo.

Por ejemplo: M8E_MODELMATTYPE (terreno , EMT_SOLID);

M8E_SETTEXTFLAG (int flag , int state)

Activa o desactiva un flag de carga de texturas que afectarán a todas las texturas cargadas a partir de su llamada. El estado puede ser TRUE o FALSE. Los flags deben ser uno de los siguientes valores CONST:

```
ETCF_ALWAYS_16_BIT = 1h;          // Cargará todas las texturas en 16 bits
ETCF_ALWAYS_32_BIT = 2h;          // Cargará todas las texturas en 32 bits
ETCF_OPTIMIZED_FOR_QUALITY = 4h;  // Optimizará la carga para mayor calidad
ETCF_OPTIMIZED_FOR_SPEED = 8h;    // Optimizará la carga para mayor eficiencia
ETCF_CREATE_MIP_MAPS = 10h;       // Creará mip maps intermedios para evitar pixelado
ETCF_NO_ALPHA_CHANNEL = 20h;      // Ignorará el canal alpha en texturas
ETCF_ALLOW_NON_POWER_2= 40h;      // Permitirá cargar texturas de cualquier resolución
```

Es recomendable activar al inicio de la ejecución aquellos flags que se consideren necesarios ya sea para conseguir mayor calidad a costa de la eficiencia o viceversa.

Por ejemplo: M8E_SETTEXTFLAG (ETCF_OPTIMIZED_FOR_QUALITY , TRUE) optimizará la carga de texturas para que sean renderizadas con la mayor calidad posible.

Funciones de interfaz gráfica de usuario (GUI)

int M8E_GUIADDFONT (string filepath)

Carga una fuente cuya ruta de fichero es dado como parámetro, solamente soporta fuentes en formato de imagen que pueden ser generadas a partir de una fuente .ttf del sistema mediante el generador de fuentes de Irrlicht. Si se desea utilizar la fuente por defecto del sistema es posible pasar como parámetro una string vacía (""). La función devuelve el identificador de la fuente cargada que puede ser usado en la siguientes funciones. Nótese que sin un identificador de fuente cargada las siguientes funciones causarían un error de ejecución.

int M8E_GUIADDTEXT (int font , string text, int x , int y , int alpha , int red , int green , int blue)

Utiliza el identificador de la fuente pasada como primer parámetro para escribir el texto dado como segundo parámetro en la pantalla. La posición de pantalla viene dada por los parámetros x, y y el color del texto es dado por los últimos parámetros alpha, red, green, blue. Devuelve un identificador de texto válido que puede ser usado posteriormente para modificar el texto que se muestra usando M8E_GUISETTEXT. Es necesario que el texto que se escribe la primera vez tenga una longitud suficiente para albergar cualquier texto que pueda recaer sobre este mismo identificador, frecuentemente se inicializa con un texto del tipo: "XXXXXXXXXX" de longitud suficiente, que luego se modifica con M8E_GUISETTEXT.

int M8E_GUISETTEXT (int textid, string texto)

Establece sobre el identificador de texto pasado como primer parámetro el texto dado como segundo parámetro. Es necesario que el identificador de texto haya sido previamente creado con M8E_GUIADDTEXT y que el texto que almacenaba inicialmente tenga suficiente longitud como para albergar el nuevo texto.

int M8E_GUIADDIMAGE (int image , int x , int y , int alpha)

Establece una imagen en la GUI, la imagen ha debido ser cargada previamente con M8E_LOADTEXTURE. Se indica su posición y finalmente un booleano que indica con TRUE/FALSE si la información de transparencia de la textura debe interpretarse o no. Devuelve un identificador válido de imagen de la GUI.

int M8E_GUISETIMAGE (int gui_id , int texture)

Modifica la textura asociada a una imagen de la GUI, el primer parámetro es el identificador de la imagen añadida a la GUI y el segundo parámetro es el identificador de la nueva textura que quiere establecerse.

int M8E_GUIMOVE (int gui_id , int x , int y)

Mueve una imagen de la GUI, el primer parámetro es el identificador de la imagen añadida a la GUI y los siguientes parámetros indican la nueva posición.

int M8E_GUIGETRECT (INTEGER , _Trect2D POINTER)

Genera un rectángulo en pantalla a partir de un identificador de elemento gráfico de GUI cargado previamente con M8E_GUIADDIMAGE ( ). El parámetro _Trect2D se pasa a través de un puntero y determina la forma del rectángulo. El tipo _Trect2D tiene los campos UpperLeftCorner y LowerRightCorner, que a su vez tienen los campos x, y para determinar mediante 2 puntos la forma del rectángulo. Las posteriores modificaciones del tamaño del rectángulo generado deben hacerse utilizando M8E_GUISETRECT utilizando el mismo elemento gráfico de GUI y el mismo puntero _Trect2D como parámetro. El valor de devolución puede ignorarse.

int M8E_GUISETRECT (INTEGER , _Trect2D POINTER)

Modifica el aspecto del rectángulo añadido con M8E_GUIGETRECT ( ). Para hacer referencia al rectángulo generado previamente basta con utilizar los mismos parámetros que se utilizaron en la primera llamada a M8E_GUIGETRECT ( ). El valor de devolución puede ignorarse.

int M8E_GUICOL (int gui_id1 , int gui_id2)

Detecta si existe colisión entre 2 elementos gráficos de la GUI dados como parámetros, devolviendo TRUE/FALSE. Esta función sólo interpreta elementos gráficos, por tanto no es posible detectar colisiones entre elementos textuales o entre un texto y un gráfico.

int M8E_GUIADDFADER ( )

Crea un identificador válido de fader para la GUI y lo devuelve como parámetro para ser usado pos las siguientes funciones.

int M8E_GUIFADERINSET (int fader_id , int alpha , int r , int g , int b , int time)

Crea un fader en la pantalla. En primer lugar debe especificarse el identificador creado con M8E_GUIADDFADER ( ), a continuación se especifican los valores de alpha rgb para el color del fader y finalmente el tiempo de duración en milisegundos.

int M8E_GUIFADEROUTSET (integer , integer , integer , integer , integer , integer)

int M8E_GUIFADERREADY (int fader_id)

Determina si el identificador de fader dado como parámetro ha terminado de realizar el fundido.

M8E_GUIREMOVE (int Gui_Element)

Elimina un elemento de la GUI cuyo identificador es dado como parámetro. Sirve para texto 3D, Imágenes 2D, etc.

int M8E_GUIREMOVEALL ( )

Elimina todo elemento activo de la GUI. En las primeras versiones de 2010 provoca un error de ejecución, así que es aconsejable utilizar en su lugar M8E_GUIREMOVE y especificar los elementos a eliminar uno por uno.

Funciones de emisores de partículas

int M8E_PARTICLEADDBOXEMITTER (float x0 , float y0 , float z0 , float x1 , float y1 , float z1 , float offx , float offy , float offz , int maxps , int minps , int mina , int minr , int ming , int minb , int maxa , int maxr , int maxg , int maxb , int minlife , int maxlife , int desv)

Añade un emisor de partículas con geometría de caja y devuelve su identificador.

x0, y0, z0 - Indican el punto de inicio de la caja emisora
x1, y1, z1 - indican el punto de fin, esto quiere decir que la caja se determina dando 2 vértices opuestos de la misma.
offx, offy, offz -Indica de forma unitaria la dirección de movimiento de las partículas emitidas.
maxps, minps - Máximo y mínimo de particulas emitidas por segundo.
mina, minr, ming, minb, maxa, maxr, maxg, maxb - Alpha rgb mínimo y alpha rgb máximo de las partículas emitidas. El color sólo se aplica si no hay ninguna textura aplicada al emisor.
minlife, maxlife - Tiempo de vida máximo y mínimo de cada partícula emitida, en milisegundos.
desv - Máxima desviación de las partículas respecto de la dirección indicada con offx, offy, offz.

int M8E_PARTICLEADDPOINTEMITTER (float x0 , float y0 , float z0 , int minps , int maxps , int mina , int minr , int ming , int minb , int maxa , int maxr , int maxg , int maxb , int minlife , int maxlife , int desv)

Añade un emisor de partículas puntual y devuelve su identificador.

x0, y0, z0 - Indican la dirección de las partículas emitidas.
maxps, minps - Máximo y mínimo de particulas emitidas por segundo.
mina, minr, ming, minb, maxa, maxr, maxg, maxb - Alpha rgb mínimo y alpha rgb máximo de las partículas emitidas. El color sólo se aplica si no hay ninguna textura aplicada al emisor.
minlife, maxlife - Tiempo de vida máximo y mínimo de cada partícula emitida, en milisegundos.
desv - Máxima desviación de las partículas respecto de la dirección indicada con x0, y0, z0.

Nótese que la posición del emisor no se indica en ninguno de los parámetros, por lo que debe ser posicionado con M8E_POSMODEL ( ).

int M8E_PARTICLEADDCYLINDEREMITTER (float , float , float , float , float , float , float , integer , float , float , float , integer , integer , integer , integer , integer , integer , integer , integer , integer , integer , integer , integer , integer)

int M8E_PARTICLEADDRINGEMITTER (float , float , float , float , float , float , float , integer , integer , integer , integer , integer , integer , integer , integer , integer , integer , integer , integer , integer , integer)

int M8E_PARTICLEADDSPHEREEMITTER (float , float , float , float , float , float , float , integer , integer , integer , integer , integer , integer , integer , integer , integer , integer , integer , integer , integer)

int M8E_PARTICLEADDFADEOUTAFFECTOR (integer , integer , integer , integer , integer , integer)

int M8E_PARTICLEADDGRAVITYAFFECTOR (integer , float , float , float , integer)

Añade un afector gravitatorio al identificador del emisor de partículas dado como parámetro. Se debe indicar el valor de la gravedad en los 3 ejes y finalmente el tiempo en milisegundos que transcurre hasta que el afector deja de tener efecto sobre la partícula emitida.

int M8E_PARTICLEADDATTRACTIONAFFECTOR (integer, float, float , float , float , integer , integer , integer , integer)

int M8E_PARTICLEADDROTATIONAFFECTOR (integer , float , float , float , float , float , float)

int M8E_PARTICLESETSIZE (int emitter_id , float x , float y)

Establece el tamaño de las partículas del emisor cuyo identificador es dado como parámetro. Los últimos parámetros indican el tamaño horizontal y vertical, respectivamente.

int M8E_ADDSHADER (string , string , integer , string , string , integer , integer)

int M8E_AIADDFOV(INTEGER, FLOAT, FLOAT, FLOAT, INTEGER, INTEGER, INTEGER, INTEGER)

Añade un campo de visión a un modelo que se situará frente a él. Es utilizado para inteligencia artificial y todavía no ha sido testeado a fondo en las primeras versiones de 2010.

Funciones del motor de física

M8E_PHYSADDMODEL (_PhyModel phys_model)

Añade el modelo físico dado como parámetro a la lista de interacciones físicas entre modelos. Los campos básicos de la estructura _PhyModel son los siguientes:

```
mass;          // Indica la masa en kilogramos asignada al modelo físico
model;         // Indica cuál es el modelo cargado que interactuará con la física
AutoUpdateDim; // Indica con TRUE o FALSE si los cambios en las dimensiones del modelo serán
               // tenidos en cuenta a la hora de resolver la física
int typemodel; // Indica el tipo de geometría de colisión del modelo, puede asignarse en base
               // a las siguientes constantes:
               //Autodetectada            TPHYS_AUTODETECT = -1
               //Esférica                 TPHYS_SPHERE = 0
               //Caja                     TPHYS_BOX = 1
               //Cilindro                 TPHYS_CYLINDER = 2
               //Cápsula                  TPHYS_CAPSULE = 3
               //Cono                     TPHYS_CONE = 4
               //Plano                    TPHYS_PLANE = 5
               //Malla convexa            TPHYS_CONVEX_MESH = 6
               //Malla cóncava (Terreno)  TPHYS_CONCAVE_MESH = 7
               //Malla cóncava de Gimpact TPHYS_CONCAVE_GIMPACT_MESH = 8
```

Otras funciones de Bullet 3D

M8E_SETSTRPARAM (string parameter , string value)

Establece parámetros de tipo string. Ejemplos:

```
m8e_setstrparam("CSM_TexturePath","media\misc"); //Directorio de las texturas cartography shop.
m8e_setstrparam("MY3D_TexturePath" ,"media\misc"); //Directorio de las texturas my3d.
```

M8E_MODELADDCHARCONTROL (int model , _CharacterControl control)

Añade un controlador animado de carácter. La estructura _CharacterControl tiene los siguientes campos:

```
float jumpForce;        // Determina el impulso aplicado al saltar.
float rotationSpeed;    // Determina la velocidad de rotación al girar.
float moveSpeed;        // Determina la velocidad de movimiento al desplazarse.

// Los siguientes valores son booleanos y generalmente se asignan a una función de detección de tecla pulsada
// dentro de un LOOP para actuar como controles.
int Forward;            // Determina si el modelo debe desplazarse hacia adelante.
int Backward;           // Determina si el modelo debe desplazarse hacia atrás.
int rotLeft;            // Determina si el modelo debe rotar hacia la izquierda (Según su eje Y).
int rotRight;           // Determina si el modelo debe rotar hacia la derecha (Según su eje Y).
int strafeLeft;         // Determina si el modelo debe desplazarse hacia la izquierda.
int strafeRight;        // Determina si el modelo debe desplazarse hacia la derecha.
int rotUp;              // Determina si el modelo debe rotar hacia arriba (Según su eje Z).
int rotDown;            // Determina si el modelo debe rotar hacia abajo (Según su eje Z).
int Crouch;             // Determina si el modelo debe agacharse.
int Jump;               // Determina si el modelo debe saltar.

// Los siguientes valores sólo funcionan con modelos animados .md2 de Quake
float AnimationSpeed;   // Determina la velocidad de animación si se trata de un  modelo animado.
int AnimationEnabled;   // Determina con TRUE/FALSE si el modelo debe realizar animación.
int frameRunStart;      // Determina el número de frame inicial de la animación de correr.
int frameRunEnd;        // Determina el número de frame final de la animación de correr.
int frameStandStart;    // Determina el número de frame inicial de la animación de quieto.
int frameStandEnd;      // Determina el número de frame final de la animación de quieto.
int frameJumpStart;     // Determina el número de frame inicial de la animación de saltar.
int frameJumpEnd;       // Determina el número de frame final de la animación de saltar.
int frameCrouchStart;   // Determina el número de frame inicial de la animación de agachado.
int frameCrouchEnd;     // Determina el número de frame final de la animaicón de agachado.

float FixRotAngle;      // Fija el ángulo de rotación en un valor determinado (No testeado).
```

Int M8E_LOADSCN (string filepath)

Carga una escena creada con irrEdit (http://irredit.irrlicht3d.org), devuelve TRUE si se cargó correctamente.

int M8E_Q3MAPQUERY (int mapModel , string EntityName , string EntityGroup , string VarName , _Q3ItemQuery result)

Pone en cola una entidad de mapa de Quake 3. Devuelve TRUE si se pone en cola correctamente. Por ejemplo: M8E_Q3MAPQUERY (mapa , "info_player_deathmatch" , "origin" , "angle" , query);

Int M8E_ADDBOARD (float positionX , float y , float z , float width , float height)

Crea un modelo de tablero con cierta posición, anchura y largura. Devuelve un identificador de modelo válido. Útil para trabajar con grids, para juegos de puzzle o de estrategia. Devuelve un identificador de tablero válido.

M8E_BOARDSIZE (int Billboard , float width , float height)

Establece la anchura y largura de un modelo de tablero pasado como parámetro.

Buenos modelos 3D y herramientas de modelado:

(Editor de niveles)
Quark
http://quark.planetquake.gamespy.com/

Qoole Q3radiant
www.planetquake.com
www.telefragged.com
www.qeradiant.com

DeleD 3D Editor
www.delgine.com

(Conversor/Visor 3D)
Deep Exploration
www.righthemisphere.com

(Editor de modelos)
MilkShape3D
www.milkshape3d.com

Anim8or
http://www.anim8or.com

Blender
http://blender.org/

Gmax:
http://www.turbosquid.com/gmax

Editor de arboles y bosques:
http://www.frecle.net/treed

Zmodeler:
http://www.zmodeler.com/

Modelos gratuitos:
www.polycount.com

Otras herramientas:
http://www.blitzbasic.com/toolbox/toolbox.php

(Excelentes herramientas 2D)
Inkscape
http://inkscape.org/

Gimp

http://www.gimp.org

Enlaces relacionados:

Fenix / Bennu

http://forum.divsite.net
http://fenixworld.se32.com
http://www.booleansoup.org
http://div.france.online.fr

Diseño de Videojuegos

http://www.stratos-ad.com
http://www.codepixel.com
http://www.gamedev.net

Programación

http://ayudaprogramacion.net

Enlaces de visita obligatoria:

---

http://www.bennugd.org
http://forum.bennugd.org
http://3dm8ee.blogspot.com
http://bennupack.blogspot.com
http://trinit.es