Generación de un sistema de animación

PID_00249854

Jorge Arnal Montoya

Tiempo mínimo de dedicación recomendado: 2 horas

Universitat Oberta de Catalunya

Índice

Introducción

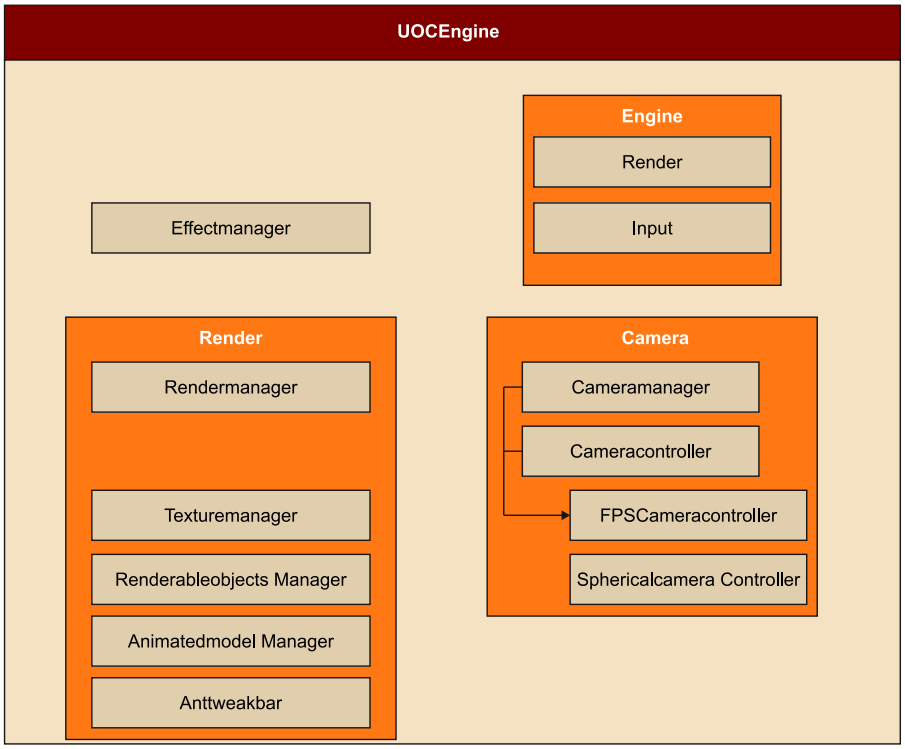
En este material, vamos a modificar nuestra arquitectura de motor de videojuegos para poder integrar dos bibliotecas creadas por terceros.

La primera biblioteca que vamos a integrar va a ser una biblioteca de animación esqueletal como Cal3D, que nos permitirá renderizar modelos animados en nuestro videojuego.

A continuación, integraremos una biblioteca como AntTweakBar, con la que podremos presentar interfaces gráficas de usuario sencillas con las que modificar las propiedades de nuestro videojuego o motor a través de una interfaz simple.

1. Arquitectura del motor

En este material, vamos a modificar la arquitectura del motor para integrar modelos animados y una biblioteca que nos permitirá mostrar y modificar las propiedades de nuestro juego mediante una interfaz de *debug*.

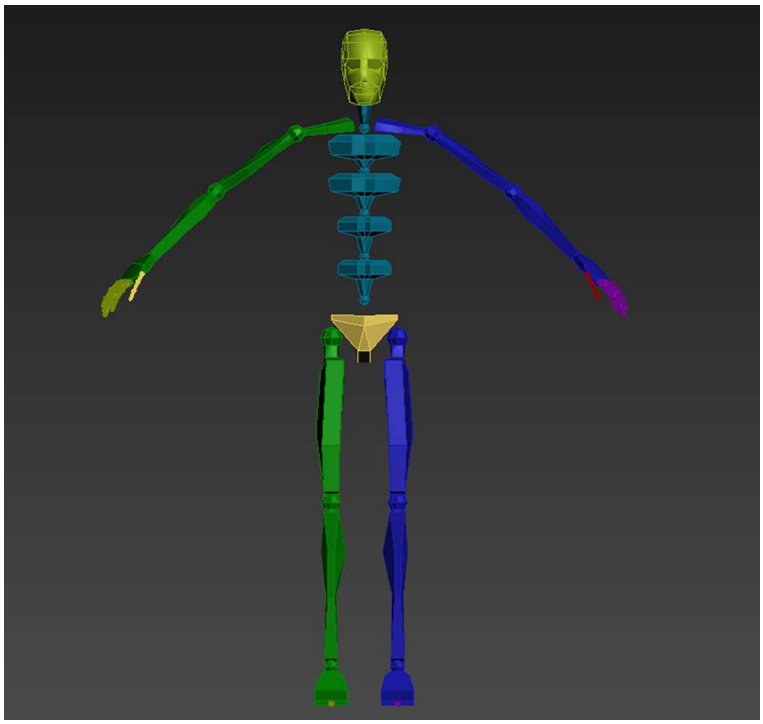


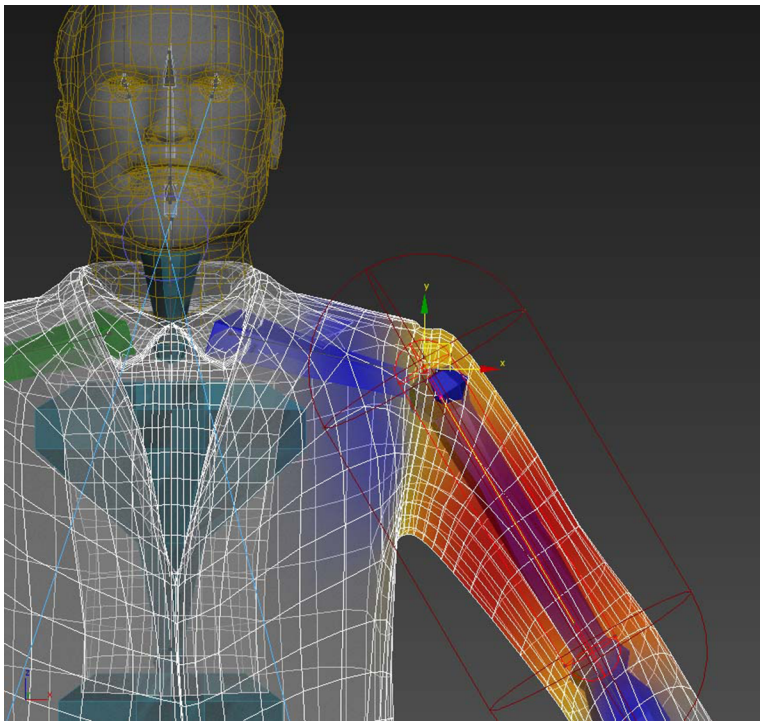
2. Animación esqueletal

En los primeros videojuegos en 3D, los modelos de los personajes eran representados mediante planos que se enfocaban siempre a cámara. Wolfenstein 3D, Doom o Duke Nukem 3D son algunos ejemplos de estos videojuegos.

Posteriormente, empezaron a llegar los primeros videojuegos que utilizaban modelos poligonales para representar a estos personajes. Estos primeros modelos animados empleaban una animación basada en la animación tradicional, donde teníamos para cada uno de los *frames* de la animación una captura del modelo 3D estático, similar a la técnica utilizada en la animación clásica 2D.

Es con la llegada de una nueva generación de videojuegos cuando aparecen los primeros modelos animados utilizando la técnica de la animación esqueletal. Este procedimiento consiste en crear un sistema de huesos o esqueleto, en el que a cada uno de los vértices se le asocia uno o más de los huesos, y se les da un peso. Este proceso se denomina *rigging* del modelo.





Una vez que tenemos el modelo pesado, se animan los huesos, que al modificarse, transforman las posiciones de los vértices según el *rigging* realizado.

La diferencia con respecto a la animación clásica es que el peso de los datos de la animación es muy inferior, ya que solo guardamos la información de transformación de los huesos por cada *keyframe*. Uno de sus inconvenientes es que requiere que cada *frame* se calcule a partir de la posición de los vértices, y esto puede ser costoso si se hace en la CPU. Hoy en día podemos programar la GPU, que nos permite realizar esta tarea directamente en ella haciendo que el proceso sea extremadamente rápido si lo comparamos con el mismo proceso realizado por la CPU.

El cálculo que se realiza en cada *frame* es para cada uno de los vértices; se multiplica su posición por cada una de las matrices de transformación de los huesos asociados por su peso ligado a ese hueso. El sumatorio total de ese cálculo será la posición del vértice transformado según la animación.

2.1. Cal3D

Dentro del sector del videojuego, se utilizan muchas bibliotecas creadas por terceros, como las API gráficas o bibliotecas que nos dan funcionalidad. En este caso vamos a utilizar una biblioteca llamada Cal3D, que nos va a permitir integrar animación esqueletal.

Esta biblioteca ha sido utilizada en infinidad de videojuegos que podemos encontrar en el mercado.

Cal3D es una biblioteca programada en C++ y que puede ser utilizada de forma independiente en cualquier plataforma.

Para la inserción de modelos animados en nuestro motor, hemos creado diferentes clases que encapsulan el código de Cal3D.

2.1.1. AnimatedModelManager

La primera clase que hemos creado para encapsular la biblioteca Cal3D es CAnimatedModelManager. Esta clase se va a comportar a modo de *manager* y controlará todos los modelos animados de nuestro motor.

Esta clase se basa, al igual que el *manager* de texturas, en un *map* de la biblioteca STL, donde almacenamos los AnimatedCoreModel según el nombre de este.

Elemento *core*

Un elemento *core* es aquel que tenemos una única vez en memoria y cuya información se utiliza por varias instancias. En el modelo animado, la información de cómo está formado el esqueleto será información *core*, y el estado de un *frame* para un modelo animado será información de instancia.

Destacamos el método GetActor, al que llamaremos cada vez que queramos recoger un AnimatedCoreModel. Este método busca dentro del *map* si encuentra ese modelo. En caso de no encontrarlo, crea el modelo, lo almacena dentro del mapa y devuelve una referencia a ese modelo; en caso de que lo encontrase dentro del mapa, devuelve una referencia sin volver a cargar el modelo.

2.1.2. AnimatedCoreModel

La clase CAnimatedCoreModel encapsula la clase calCoreModel de Cal3D. En esta clase guardaremos toda la información *core* de nuestro modelo animado. Esta información *core* es aquella que se repite para todas las instancias de los modelos animados.

Esta clase derivará de la clase CXMLParser, que nos permitirá leer un fichero XML, donde encontraremos toda la información que define nuestro modelo animado.

En esta clase remarcaremos los siguientes métodos:

- LoadMesh. Este método carga la información de malla de nuestro modelo animado, información almacenada en un fichero de tipo CMF de Cal3D.
- LoadSkeleton. Este método carga la información de esqueleto de nuestro modelo animado, información almacenada en un fichero de tipo CSF de Cal3D.

- LoadAnimation. Este método carga la información de una animación de nuestro modelo animado, información almacenada en un fichero de tipo CAF de Cal3D.

2.1.3. AnimatedInstanceModel

La última clase que vamos a utilizar para encapsular los modelos animados de Cal3D será CAnimatedInstanceModel, que va a derivar de la clase CRenderableObject, la cual nos permite pintar un elemento de nuestro juego.

Esta clase encapsulará la clase calModel de Cal3D, que contiene la información de instancia de nuestro modelo animado. También utilizará la clase calHardwareModel de Cal3D para poder utilizar un *shader* para el pintado del modelo animado. Gracias a este *shader*, los cálculos de transformación de cada uno de los vértices del modelo animado se realizarán en la GPU.

De esta clase destacaremos los siguientes métodos:

- Initialize. Este método construirá los objetos de Cal3D calModel y calHardwareModel. A continuación, adjuntará al modelo de calModel todas las mallas que contiene el modelo. Además, ejecutará la primera animación en el modelo y hará una actualización de 0 para que el modelo salga en el primer *frame* con la pose del modelo animado.
- LoadVertexBuffer. Este método será el responsable de crear el VertexBuffer y el IndexBuffer que definen nuestro modelo animado. Para ello utilizará la clase calHardwareModel de Cal3D, donde deberemos determinar el formato del vértice para acabar llamando al método Load, que rellenará en nuestro VertexBuffer la información de los vértices.
- Update. Este método deberemos llamarlo en cada *frame* y actualizará el modelo animado según el ElapsedTime.
- Render. El método Render realizará el pintado de nuestro modelo animado. Para ello, recorrerá todas las submallas que componen nuestro HardwareModel, establecerá las constantes de las matrices de transformación de los huesos que componen esta submalla y llamará al método DrawIndexed de la clase CRenderableVertex.
- ExecuteAction. Con este método, ejecutaremos una animación de tipo acción en nuestro modelo animado.
- BlendCycle. Con este método, ejecutaremos una animación de tipo ciclo en nuestro modelo animado.
- ClearCycle. Con este método, eliminaremos una animación de tipo ciclo del *mixer* del modelo animado.

Tipos de animaciones en Cal3D

En Cal3D diferenciamos entre dos tipos de animaciones:

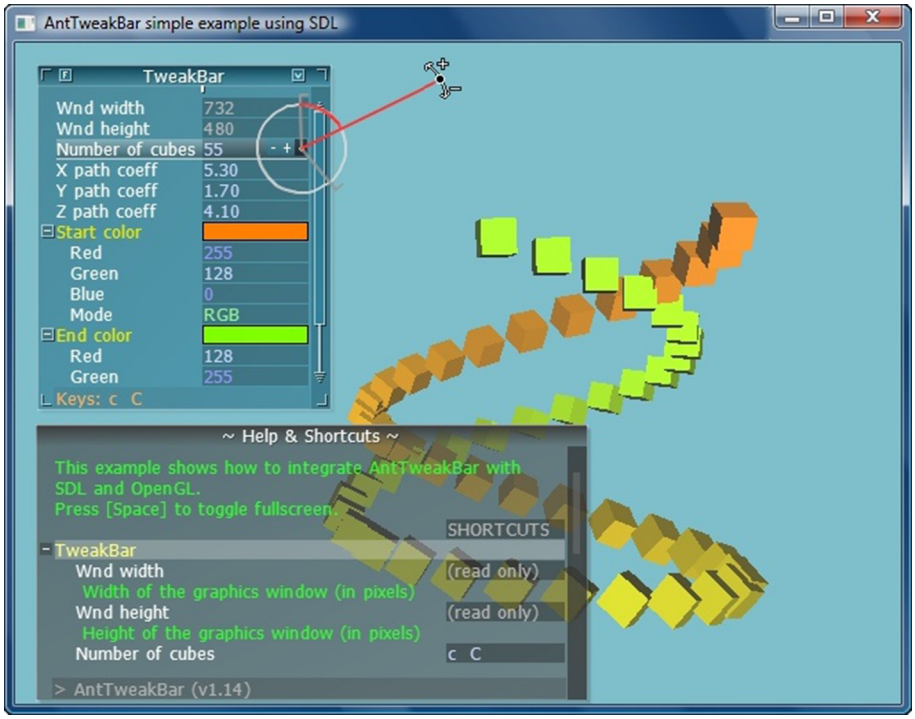
1) Ciclo. Es una animación que una vez que termina vuelve a empezar interrumpidamente (por ejemplo, la animación de caminar, *idle*…).

2) Acción. Es una animación que una vez que termina se quita del *mixer* de animaciones (por ejemplo, la animación de golpear, disparar…).

3. AntTweakBar

Al igual que hemos utilizado la biblioteca Cal3D para integrar la funcionalidad de animación esqueletal, vamos a integrar una biblioteca como AntTweakBar, que nos permitirá crear interfaz gráfica en modo *debug* para poder modificar los parámetros de nuestro motor.

La biblioteca AntTweakBar permite integrarse con API gráficas basadas en OpenGL, Directx 9 y DirectX 11.



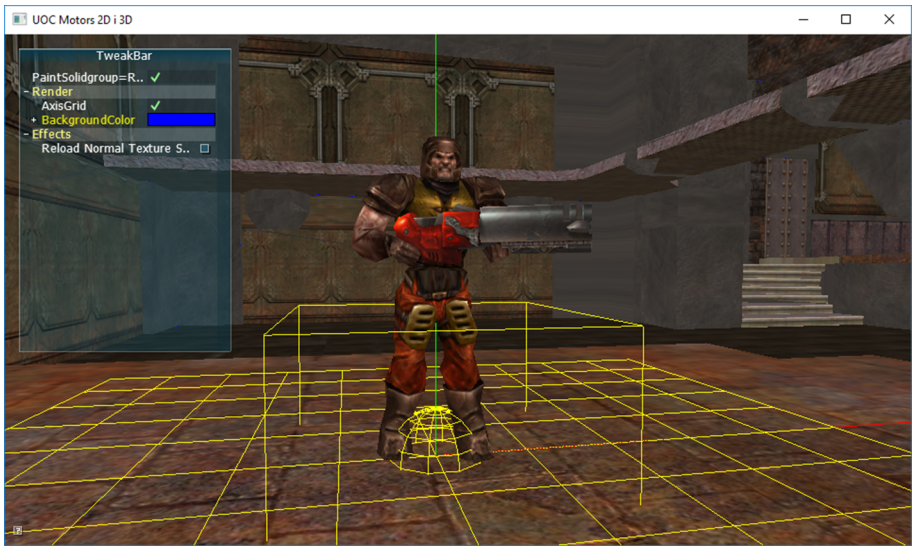
Una biblioteca como esta nos va a permitir crear interfaces que nos pueden facilitar mucho el trabajo en desarrollo, ya que, a través de sus diferentes controladores, podremos modificar las propiedades de nuestro videojuego en tiempo real.

Para su integración, hemos realizado los siguientes pasos:

- Añadir en la clase CUOCEngine un objeto de la clase TwBar.
- En el método Init de la clase CUOCEngine, construir nuestra TwBar añadiendo aquellos parámetros que queremos que sean modificables a través de la interfaz gráfica.
- En el método Render, llamar a la función TwDraw después del pintado de la escena.
- En el destructor de la clase, llamar a la función TwTerminate para eliminar las referencias y los objetos de memoria de la biblioteca.

- En el MsgProc de la clase CApplicationDX, añadir la llamada a la función TwEventWin.

En nuestro motor de juegos, el resultado de integrar una biblioteca como AntTweakBar será el siguiente.



4. Retos

En este material vamos a realizar los siguientes retos:

- Importar de fichero los elementos que definen el modelo animado 3D, estos son:
  - Esqueleto
  - Malla
  - Animaciones

- Renderizar el modelo 3D animado.
- Integración de una shader para animar el modelo mediante la GPU.

Resumen

En este material, hemos aprendido que en la industria del videojuego se utilizan diferentes bibliotecas que facilitan el desarrollo de los juegos.

Hemos empezado aprendiendo los fundamentos de la animación esqueletal y hemos integrado una biblioteca como Cal3D, que nos facilita la inserción de modelos animados dentro de nuestro motor de juegos.

A continuación, hemos conocido una biblioteca como AntTweakBar, que nos permite crear interfaces gráficas de usuario sencillas con las que poder modificar las propiedades y las características de nuestro motor de videojuegos en tiempo real.

Bibliografía

Cal3D 3D Character Animation Library. http://home.gna.org/cal3d/

Beginner's Guide to 3ds Max - 11. Rigging and Skinning. https://www.youtube.com/watch?v=XVB9W6xhBcs

AntTweakBar. http://anttweakbar.sourceforge.net/doc/