ANIMACIÓN ESQUELETAL

Durante los próximos días aprenderemos todo lo necesario para exportar modelos animados desde el 3D Studio MAX hasta nuestro videojuego.

Introducción a la animación en videojuegos

Desde el principio, los personajes de los videojuegos han intentado estar más o menos animados, al principio en los videojuegos en 2D se utilizaban técnicas utilizadas en industrias como el cine o películas de animación tradicional.

Con la llegada de las 3D los personajes dejan de ser *sprites* para ser modelos en 3D, dando una nueva problemática.

La primera técnica que se utilizó para personajes 3D es la conocida como animación por vértice. Esta técnica consistía al igual que en 2D tenemos por cada frame una captura de nuestro personaje, hacemos una captura de nuestro personaje en 3D por cada uno de los frames. Esta técnica tiene diferentes ventajas:

- es relativamente fácil de implementar
- tiene poca carga de CPU, ya que la animación está precalculada

pero también tiene sus desventajas:

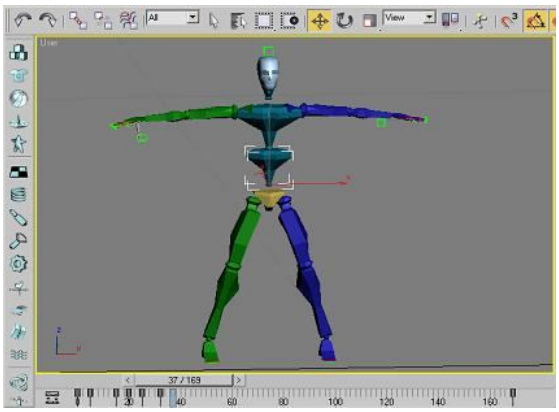
- a medida que aumenta el número de polígonos, aumenta exponencialmente el peso de los datos
- puede dar animaciones poco fluidas ya que las animaciones están capturadas en frames determinados, para frames intermedios no tenemos animación y por tanto no es fluido además el blend de dos animaciones no es demasiado correcto

A medida que los ordenadores y las tarjetas de video se vuelven más potentes, las mallas se hacen más pesadas (con mayor número de polígonos), por tanto deja de ser interesante utilizar la animación de estas mallas con el método de animación por vértice. A raíz de ello aparece el método conocido como animación esqueletal.

¿Qué es la animación esqueletal?

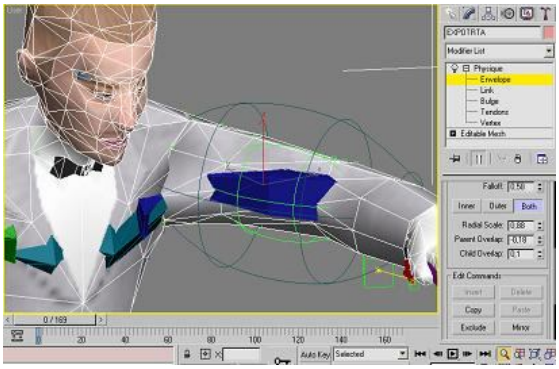
La animación esqueletal se basa en simular la animación de un modelo 3D a través de la animación de unos huesos

Partiendo de un esqueleto formado por huesos.



Esqueleto

Tenemos una malla de vértices que se le aplican unos pesos a cada uno de los vértices asignando unos huesos, lo que se conoce como rigging.



Malla aplicándole los pesos a los vértices

Una vez realizado este paso, animamos los huesos del esqueleto y con las matrices de transformación resultantes las multiplicamos por los vértices y los pesos dándonos la malla animada.

Malla con animación esqueletal

CAL3D

Implementar la animación esqueletal puede ser una faena bastante complicada y por consecuencia bastante costosa. Como explicamos en las primeras sesiones el mundo de los videojuegos es una industria y como tal muchas veces se utilizan herramientas o componentes desarrollados de forma externa o por terceras empresas, uno de los casos típicos de componentes utilizado en los videojuegos es la parte de la animación.

En nuestro proyecto utilizaremos un componente de uso profesional como es el Cal3D.

Descarga del CAL3D

Para trabajar directamente con la librería de Cal3D lo que haremos será descargar la librería directamente de la web del proyecto Cal3D.

http://gna.org/projects/cal3d/

En dicha web accediendo a la sección de descarga accedemos a la sección de source, dónde nos encontramos diferentes versiones de la librería además de diferentes ejemplos y utilidades.

Para empezar descargaremos la librería en su versión 0.11 rc2 la versión completa con código fuente.

Una vez tenemos descargada dicha versión y descomprimida nos encontramos con una estructura de directorio dónde encontramos:

- *data*, carpeta dónde encontraremos diferentes modelos exportados del 3D Studio MAX de ejemplo
- *docs*, documentación de la librería en formato doxygen
- *examples*, diferentes ejemplos que muestran como utilizar la librería
- *plugins*, diferentes plugins para poder exportar las mallas para la mayoría de las aplicaciones de 3D del mercado
- *src*, código fuente dónde encontramos la librería
- *tests*, ejemplo básico de animación esqueletal
- *toolspec*, herramientas de instalación

Compilar la librería

Para poder generar la librería deberemos seleccionar el archivo de proyecto del cal3d que se encuentra dentro de la carpeta *src*.

Una vez introducida dentro de nuestra solución podremos compilar esta librería, al linkar dicha librería nos avisará de que no encuentra determinados métodos de diferentes clases, habrá que añadir a mano dichos archivos cpp dentro del proyecto.

Una vez compilada correctamente la librería podremos empezar a trabajar con el cal3d, teniendo por una parte los ficheros .h, la librería .lib y la librería dinámica que se deberá adjuntar con el juego .dll.

Ejemplo Viewer

Dentro del proyecto Cal3D nos encontramos con un proyecto llamado cal3d_miniviewer_d3d. En dicho proyecto nos encontramos con los ficheros:

- *main.cpp*, código fuente dónde encontramos la creación de la aplicación y del directx
- *tick.cpp*, código fuente que devuelve el número de milisegundos transcurridos entre frames
- *viewer.cpp*, la clase principal dónde nos muestra las funcionalidades básicas del cal3D

Dentro de la clase Cviewer nos encontramos los siguientes métodos que pasaremos a detallar a continuación:

- *CViewer*, el constructor crea la instancia del objeto CalCoreModel con el nombre del core mediante el método new CalCoreModel("dummy");
- *onCreate*, método que parsea el command line que le pasamos al ejecutar la aplicación. Llama al método *parseModelConfiguration* que parsea la entrada y por último crea el model instance de cal3D con el *new CalModel(m_CalCoreModel)*
- *parseModelConfiguration*, parsea el fichero de entrada para cargar el esqueleto del modelo con el método loadCoreSkeleton, cargar la malla del modelo con el método *loadCoreMesh* y por último carga cada una de las animaciones con *loadCoreAnimation*.
- *onInit*, recorre cada una de las mallas del *CalCoreModel* y las atacha al *CalModel* mediante el método *attachMesh*. Por último introduce en el *calMixer* del *CalModel* la primera animación de tipo acción o de tipo ciclo con los métodos *executeAction* o *blendCycle*. Por último llama al método *loadVertexBuffer*.
- *loadVertexBuffer*, crea el vertex buffer y el index buffer necesario para cargar la malla de triángulos del modelo. Cuidado porque el tamaño del

vertex buffer y del index buffer no corresponde con el número de vértices y triángulos que contiene la malla.

- *onIdle*, simplemente llama al método update del calModel pasándole el número de milisegundos transcurridos en el último frame.
- *onShutDown,* destruye todo los elementos creados dinámicamente de la clase.
- *renderModel*, renderiza el modelo mediante el calRenderer, el vertex buffer y el index buffer creado previamente. Hay que fijarse que sube el array de vértice e índices cada frame.

Exportación de modelos desde el MAX

Para poder exportar modelos en el 3D Studio MAX 2010 debemos instalar el plugin de exportación del Cal3D en la carpeta de plugins.

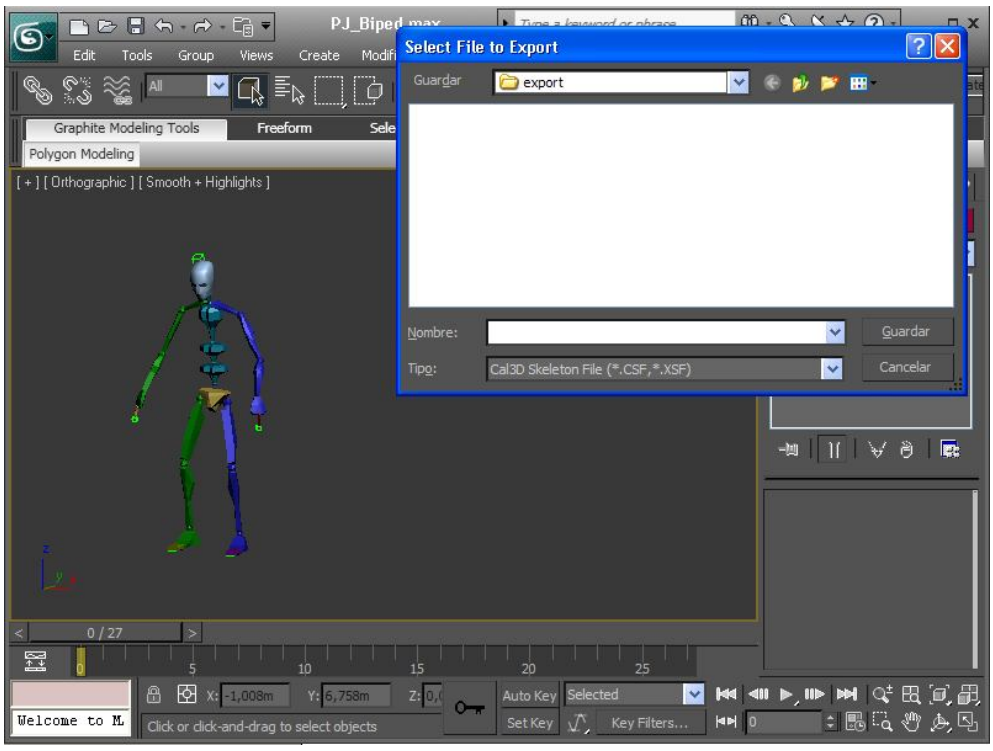
Dicho plugin ha de ser compilado previamente con el SDK del 3D Studio MAX.

Una vez instalado el plugin podremos exportar el esqueleto, las mallas y las animaciones de los modelos.

Para exportar el esqueleto desde el 3D Studio MAX deberemos elegir del esqueleto del modelo el hueso con el centro de masas.

Una vez seleccionado el hueso iremos al menú principal y daremos a la opción de exportar.

Nos saldrá una ventana como la siguiente.



Para exportar el esqueleto deberemos seleccionar la opción de *Cal3D Skeleton File (*.CSF, *.XSF)*.

Para exportar la malla seleccionaremos en el visor la malla del modelo y le daremos a la opción de exportar *Cal3D Mesh File (*.CMF, *.XMF).*

Y por último para exportar cada una de las animaciones le daremos a la opción de exportación *Cal3D Animation File (*.CAF, *.XAF)*.

Si queremos ampliar la información de como exportar un modelo o cualquier cosa como siempre deberemos ir a la documentación de la propia librería.

Tipo de animaciones

Dentro de los tipos de animación que nos encontraremos hemos de distinguir entre dos diferentes:

- *animación de tipo ciclo*, es un tipo de animación la cual la ponemos en el mixer de animaciones del cal3d y se repetirá indefinidamente. Un ejemplo sería: animación de idle, caminar, correr, ...
- *animación de tipo acción*, és un tipo de animación la cual introducimos en el mixer de animaciones y se mezclará con la animación de tipo ciclo ejecutándose una única vez. Un ejemplo sería: disparar, dar un golpe, ...

AnimatedModelsManager

Al igual que realizamos con las mallas estáticas tendremos un mánager de modelos animados que derivará de la clase CmapManager. Esta clase la implementaremos como sigue.

```
class CAnimatedModelManager : public CMapManager<CAnimatedCoreModel>
{
public:
        CAnimatedModelManager();
        ~CAnimatedModelManager();
        CAnimatedCoreModel * GetCore(const std::string &Name, const std::string &Path);
        CAnimatedInstanceModel * GetInstance(const std::string &Name);
        void Load(const std::string &Filename);
};
```

Lo que nos interesa de esta clase son los métodos:

- *GetCore*, este método se encargará de cargar un CAnimatedCoreModel en caso de no existir y nos devolver su dirección de memoria. Utilizaremos el mapa de la clase CmapManager para guardar la estructura.
- *GetInstance*, nos devolverá una instancia de tipo CAnimatedInstanceModel según el nombre de la core cargada previamente.
- *Load*, cargará un fichero xml añadiendo los elementos cores dentro del CMapManager<CAnimatedCoreModel>

AnimatedCoreModel

Esta clase encapsulará a la clase CalCoreModel del Cal3D facilitándonos el trabajo para cargar un modelo. Su estructura será la siguiente.

```
class CAnimatedCoreModel
{
private:
	CalCoreModel*                   m_CalCoreModel;
	std::string                     m_Name;
	std::vector<std::string>        m_TextureFilenameList;
	std::string                     m_Path;

	bool LoadMesh(const std::string &Filename);
	bool LoadSkeleton(const std::string &Filename);
	bool LoadAnimation(const std::string &Name, const std::string &Filename);
public:
	CAnimatedCoreModel();
	~CAnimatedCoreModel();
	CalCoreModel *GetCoreModel( );
	const std::string & GetTextureName( size_t id );
	size_t GetNumTextures( ) const { return m_TextureFilenameList.size(); }
	void Load(const std::string &Path);
	int GetAnimationId(const std::string &AnimationName) const;
};
```

De esta clase explicamos los siguientes métodos:

- *LoadMesh*, se encargará de cargar la malla del modelo animado con los métodos correspondientes del Cal3D.
- *LoadSkeleton*, se encargará de cargar el esqueleto del modelo animado.
- *LoadAnimation*, cargará la animación del modelo según el FileName.
- *GetCoreModel*, nos devuelve el modelo core del Cal3D.
- *Load*, este método cargará un fichero xml y cargará todos los elementos de este modelo a partir del path pasado como parámetro.
- *GetAnimationId*, nos devolverá el Id de la animación según su nombre dentro del CalCoreModel.

AnimatedInstanceModel

Esta clase encapsulará a la clase CalModel del Cal3D, su estructura será la siguiente.

```
class CAnimatedInstanceModel : public CRenderableObject
{
private:
	CalModel                        *m_CalModel;
	CAnimatedCoreModel              *m_AnimatedCoreModel;
	std::vector<CTexture *>         m_TextureList;
	LPDIRECT3DVERTEXBUFFER9         m_pVB;
```

```
        LPDIRECT3DINDEXBUFFER9          m_pIB;
        int                             m_NumVtxs;
        int                             m_NumFaces;

        bool LoadVertexBuffer(CRenderManager *RM);
        void LoadTextures();
public:
        CAnimatedInstanceModel();
        ~CAnimatedInstanceModel();
        void Render(CRenderManager *RM);
        void RenderModelBySoftware(CRenderManager *RM);
        void Update(float ElapsedTime);
        void Initialize(CAnimatedCoreModel *AnimatedCoreModel, CRenderManager *RM);
        void Destroy();
        void ExecuteAction(int Id, float DelayIn, float DelayOut, float WeightTarget=1.0f, bool
AutoLock=true);
        void BlendCycle(int Id, float Weight, float DelayIn);
        void ClearCycle(int Id, float DelayOut);
        bool IsCycleAnimationActive(int Id) const;
        bool IsActionAnimationActive(int Id) const;
};
```

Esta clase comentaremos los siguientes métodos:

- *Initialize*, se encargará de inicializar los datos necesarios para crear la instancia del cal3D.
- *InitD3D*, inicializará los elementos necesarios de DirectX para después poder renderizar el modelo.
- *Update*, realizará el update del modelo de Cal3D.
- *RenderModelBySoftware*, implementará la parte necesaria para renderizar el modelo vía software. Explicaremos como renderizar el model vía hardware durante las sesiones de Shaders en las próximas semanas.
- *Destroy*, destruirá todos los elementos creados en memoria de forma dinámica.
- *ExecuteAction*¸ introducirá una animación de tipo acción en el mixer del Cal3D.
- *BlendCycle*¸ introducirá una animación de tipo ciclo en el mixer del Cal3D.
- *ClearCycle*, eliminará una animación de tipo ciclo del mixer del Cal3D.
- *IsCycleAnimationActive*, devolverá si la animación de tipo ciclo está o no activa en el mixer.
- *IsActionAnimationActive*, devolverá si la animación de tipo acción está o no activa en el mixer.