3D STUDIO MAX SDK

Durante los próximos días aprenderemos todo lo necesario para crear plug-ins para 3D Studio MAX 2010.

Crear un plugin para MAXScript

Para crear nuestro primer plugin con el SDK del 3D Studio MAX nos basaremos en la ayuda en el apartado *Writing plug-ins/Getting Started*.

Lo primero que haremos será instalar el Wizard de Visual Studio 2008 según dice el apartado *Installing the Plug-in Wizard (AppWizard)*.

Una vez tenemos instalado el Wizard generaremos un nuevo plug-in de tipo *Utility*, después introduciremos la categoría y la descripción del plug-in, podemos dejarlo en blanco si lo deseeamos. En la siguiente pestaña introduciremos los paths dónde se encuentran las carpetas necesarias para poder compilar un plug-in de 3D Studio MAX.

El plug-in que acabamos de crear tiene elementos que no necesitamos, así que pasaremos a customizar las propiedades del proyecto para dejarlo únicamente con lo necesario.

- Primero borraremos el resource file, archivo .rc
- Borraremos los ficheros resource.h, PluginName.h, 3dsmaxsdk_preinclude.h
- Modificaremos el fichero DllEntry.cpp por el contenido que nos dice el apartado de la ayuda *Writing MAXScript Plug-ins/MAXScript Plug-in Basics*.

```
#include "maxscrpt/maxscrpt.h"

HINSTANCE hInstance;

// ==========================================================
// Grab onto this DLL's instance handle

BOOL WINAPI DllMain(HINSTANCE DLLhinst, DWORD fdwReason, LPVOID
lpvReserved)
{
  switch (fdwReason)
  {
            case DLL_PROCESS_ATTACH:
            hInstance = DLLhinst;
            break;
  }
  return TRUE;
}

__declspec (dllexport) void LibInit()
{
```

```
    // TODO: Put any code for initializing your plug-in here.
    MessageBox(NULL, _T("My MAXScript plug-in is loaded!!"), _T("Testing!"),
MB_OK);
}

__declspec (dllexport) const TCHAR* LibDescription()
{
    // TODO: Put code in here telling what your plug-in does.
    return _T("My MAXScript plug-in");
}

__declspec (dllexport) ULONG LibVersion()
{
    // Return the version of the Max SDK
    return VERSION_3DSMAX;
}
```

- También modificaremos el fichero .def según nos dice la ayuda *Writing MAXScript Plug-ins/MAXScript Plug-in Basics*.

```
LIBRARY PluginName
EXPORTS
    LibDescription    @1
    LibInit           @2
    LibVersion        @3
SECTIONS
.data READ WRITE
```

- Por último deberemos modificar las propiedades del proyecto, para realizar esta tarea nos basaremos en el plug-in para MAXScript testdlx que encontramos en el SDK de la ayuda de 3ds Max 2010 que encontramos en la carpeta SDK\maxsdk\howto\maxscript\testdlx.
- Recordar que el plug-in al ser de tipo MAXScript deberemos generarlo con extensión .dlx.

Lo hacemos de esta manera para no tener que comenzar de cero un proyecto y tener que definir las propiedades a mano.

Registrar funciones para MAXScript

Para poder registrar funciones implementadas en C que puedan ser utilizadas dentro del MAXScript deberemos registrarlas para hacerlas visibles, para ello utilizaremos la función *def_visible_primitive* del SDK.

Para registrar una función para mostrar el típico "*Hello World"* implementaríamos el siguiente código.

```
#include "maxscrpt/maxscrpt.h"
#include "maxscrpt/definsfn.h"

def_visible_primitive( HelloWorld, "hello_world");
```

La implementación de la función sería la siguiente:

```
Value* HelloWorld_cf(Value **arg_list, int count)
{
	::MessageBox(NULL, "Hello World", "MAX Script Function", MB_OK);
	return &ok;
}
```

Crear un plugin para exportar los vértices

Una vez sabemos publicar funciones vamos a implementar el código necesario para exportar la malla de vértices de forma más rápida.

Para ello vamos a utilizar unas funciones implementadas en C que nos van a permitir realizar la búsqueda de los vértices a introducir en el vertex buffer de forma más rápida.

Esta tarea la vamos a hacer mediante un mapa dónde la clave del mapa será el un vector de floats (el vértice) y el valor va a ser el índice dónde se encuentra en el array de vértices dicho vértice.

Este mapa deberemos tener uno para cada uno de los materiales que tenemos, por tanto tendremos un vector de mapas. La implementación sería la siguiente.

```
typedef std::map<std::vector<float>, int> MapVtxs;
std::vector<MapVtxs> g_MapVtxsByMaterial;
```

Una vez tenemos creado la estructura de datos necesaria para poder implementar el plugin introduciremos dos funciones que serán públicas para el MAXScript, dichas funciones serán las siguientes.

```
Value* UABMapClean_cf(Value** arg_list, int count)
{
	//Borraremos los elementos del mapa
	return &true_value;
}

/*
Esta función recibirá dos parámetros
1 : Array de floats que contendrá el vértice
2 : Integer que será el Id del Material
Devolverá el índice del vértice según el mapa, si no existe el vértice lo introducirá en el mapa
*/
Value* UABMapAdd_cf(Value** arg_list, int count)
{
	//Contenido de la función
	return &false_value;
}
```

Para poder implementar todo esta funcionalidad deberemos conocer las siguientes funciones del SDK del 3D Studio MAX.

```
#include "MAXScrpt/Numbers.h"
```
//Incluiremos este header del SDK para poder trabajar con los diferentes tipos que utilizaremos en las funciones, **incluirlo antes del header de #include "maxscrpt/definsfn.h"**

```
check_arg_count(UABMapAdd, 2, count);
```
//Comprueba que dicha función recibe 2 parámetros según el count que nos llega como parámetro de la función

```
type_check(arg_list[0], Array, "uab_map_add");
type_check(arg_list[1], Integer, "uab_map_add");
```
//Comprueba que el primer parámetro de la función recibe un argumento de tipo Array y el segundo de tipo Integer

```
Value *v1 = arg_list[ 0 ];
Value *v2 = arg_list[ 1 ];
```

//Se guarda el valor de los parámetros recibidos en la variable arg_list

```
int IdMaterial=v2->to_int();
```

//Convierte el segundo parámetro a entero y recoge el valor del Inmaterial

```
Array *vtx=(Array *)v1;
```
//Convierte el primer parámetro a Array

```
for(int i=0;i<vtx->size;++i)
{
	Value *l_Value=vtx->get(1+I);
	float l_ValueFloat=l_Value->to_float();
}
```
//Recorre el array recibido como parámetros y recupera todos los valores del array como float

```
return Integer::intern(value);
```
//Devuelve un valor a MAXScript de tipo Integer con valor value

Con esta documentación podemos implementar todo lo necesario para registrar funciones para ser utilizadas en MAXScript y poder exportar nuestras mallas de forma rápida.

Alternativa a crear un plugin

A continuación vamos a ver una alternativa a crear un plug-in con 3D Studio MAX para poder generar el mapa de los vértices de nuestra malla, para ello vamos a utilizar un objeto .net dentro de MAX de la clase Dictionary.

Podemos encontrar información de cómo utilizar la clase dictionary en http://msdn.microsoft.com/es-es/library/xfhwa508.aspx.

```
--Creamos un objeto de tipo dictionary de .net, dónde la clave es de tipo string y el valor es de tipo int16
```

```
local l_VtxsMap=dotnetobject
"System.Collections.Generic.Dictionary`2[System.String,System.Int16]"
```

Como podemos apreciar la clave del diccionario es de tipo string y nuestros vértices son de tipo Array de floats, lo que hacemos es convertir el array de floats a un string con la siguiente función.

```
function ConvertVtx Vtx =
(
	local l_Text = stringstream ""
	local l_Value=""
	for i=1 to Vtx.count do
	(
		format "%," Vtx[i] to:l_Text
	)
	return (l_Text as string)
)
```

Por último creamos la función AddVertex para que soporte el dictionary de .net, con los métodos ContainsKey nos dice si el diccionario contiene la Key según el string del vértice, con el método Add introducimos una elemento clave-valor en el diccionario y con la propiedad Item podemos buscar un valor dentro del diccionario según la clave.

```
function DotNetAddVertex Vtxs FullVtx VtxMap =
(
	local l_VtxString=ConvertVtx FullVtx
	local l_Idx=1
	if (VtxMap.ContainsKey (l_VtxString))==false then
	(
		append vtxs FullVtx
		l_Idx=vtxs.count-1
		VtxMap.Add l_VtxString l_Idx
	)
	else
		l_Idx=VtxMap.Item[l_VtxString]
	return  l_Idx
)
```