Level Editor

library:

● DirectXcollision.h ● DirectXMath.h ● DirectXtex.h ● d3d11.h ● d3dcompiler.h ● window.h ● Winbase.h ● Windowsx.h	● fbxsdk.h ● lua.hpp ● luabridge.h ● vector ● sstream ● stdlib ● memory

DirectX Lib(DirectXMath.h, DirectXcollision, DirectXtex.h, d3d11.h, d3dcompiler.h):

DirectX ?

Microsoft DirectX is a collection of application programming interfaces (APIs) for handling tasks related to multimedia, especially game programming and video, on Microsoft platforms.

- The DirectXMath Library implements an optimal and portable interface for arithmetic and linear algebra operations on single-precision floating-point vectors (2D, 3D, and 4D) or matrices (3×3 and 4×4). The library has some limited support for integer vector operations. These operations are used extensively in rendering and animation by graphics programs.

- The DirectXTex library includes a full-featured DDS reader and writer including legacy format conversions, a TGA reader and writer, a WIC-based bitmap reader and writer (BMP, JPEG, PNG, TIFF, and HD Photo), and various texture processing functions. This is intended primarily for tool usage.

- The DirectXcollision library is a DirectXMath's library.

- The d3dcompiler library is Compile HLSL code or an effect file into bytecode for a given target.

How does it work ?

Graphics.h	Graphics.cpp

class GraphicsClass
{
public:
GraphicsClass();
GraphicsClass(const GraphicsClass&);
~GraphicsClass();
 	
bool Initialize(int, int, HWND);
void Shutdown();
bool Frame();
 	
private:
bool Render();
D3DClass* m_D3D;
};
 	
#endif
	bool GraphicsClass::Initialize(int screenWidth, int screenHeight, HWND hwnd)
	{
	bool result;
	// Create the Direct3D object.
	m_D3D = new D3DClass;
	if(!m_D3D)
	{
	return false;
	}
	// Initialize the Direct3D object.
	result = m_D3D->Initialize(screenWidth, screenHeight, VSYNC_ENABLED, hwnd, FULL_SCREEN, SCREEN_DEPTH, SCREEN_NEAR);
	if(!result)
	{
	MessageBox(hwnd, L"Could not initialize Direct3D", L"Error", MB_OK);
	return false;
	}
	return true;
	}

Here we create the D3DClass object and then call the D3DClass Initialize function. We send this function the screen width, screen height, handle to the window, and the four global variables from the Graphicsclass.h file. The D3DClass will use all these variables to setup the Direct3D system.

Windows lib(window.h, Winbase.h, Windowsx.h)

Windows lib ?

it’s a Windows-specific header file for the C programming language which contains declarations for all of the functions in the Windows API, all the common macros used by Windows programmers, and all the data types used by the various functions and subsystems. It defines a very large number of Windows specific functions that can be used in C. The Win32 API can be added to a C programming project by including the <windows.h> header file and linking to the appropriate libraries. To use functions in *xxxx*.dll, the program must be linked to *xxxx*.lib (or lib*xxxx*.dll.a in MinGW). Some headers are not associated with a .dll but with a static library.

How does it work ?

CreateWindow:

```
HWND CreateWindow(
    LPCTSTR lpClassName,            // Pointeur sur une classe de fenêtre.
    LPCTSTR lpWindowName,           // Pointeur sur le texte de la fenêtre.
    DWORD dwStyle,                  // Style de la fenêtre.
    int x,                          // Position horizontale de la fenêtre.
    int y,                          // Position verticale de la fenêtre.
    int nWidth,                     // Largeur de la fenêtre.
    int nHeight,                    // Hauteur de la fenêtre.
    HWND hWndParent,                // Handle de la fenêtre parent.
    HMENU hMenu,                    // Handle de menu ou ID de contrôle.
    HANDLE hInstance,           // Handle d'instance de l'application.
    LPVOID lpParam                  // Pointeur sur des données passées à WM_CREATE.
   );
```

Exemple:

```
HWND hwnd;

   hwnd = CreateWindow("MaWinClass", "Titre", WS_OVERLAPPEDWINDOW,
                       CW_USEDEFAULT, CW_USEDEFAULT, 400, 300,
                               NULL, NULL, hinstance, NULL);
```

fbxsdk lib:

FBX ?

Yup, it mean “**filmbox**”. At the beginning, the fbx file format was designed to support the capturing motion of motion data for a software called filmbox. We called it “**FBX**” because microsoft extension filename are limited to 3 character.

How does it work ?

Well, let’s first take a look to another file format simpler than FBX which is the **OBJ file format**.
It is one of the easiest file format, from a programmer’s point of view. Here is a snapshot of an OBJ file.

```
# cube.obj
```

```
#

o cube

v  0.0  0.0  0.0
v  0.0  0.0  1.0
v  0.0  1.0  0.0
...

vn  0.0  0.0  1.0
vn  0.0  0.0 -1.0
vn  0.0  1.0  0.0
...

vt 0.25 0.0
vt 0.5  0.0
vt 0    0.25
...

f  1/11/2  7/14/2  5/12/2
f  1/11/2  3/13/2  7/14/2
f  1/7/6  4/4/6  3/3/6
...
```

- The **v** stand for vertices (x, y, z, [w])
- The **vt** stand for texture coordinate (u, v, [w])
- The **vn** stand for normals (x, y, z)
- The **f** stand for faces, they are indices.

Ok, but you doesn't answer the question

Here we go. The OBJ format is fairly simple, but we have a problem, how could we make **animation**, or **squeleton** ? Well, we can't.
FBX allow us to make this kind of cool stuff. But it is a bit more complicated than OBJ files.

Let's first take a look to a logical viewport.
- All data are represented by a **Scene**. This scene **is the world**. *see appendix 1*
- There is one **root node** in the scene. And this node can have multiple **child node**.
- A node can have information **attached** to him. Information like **Mesh**, **Camera** or **Light**. *see appendix 2*

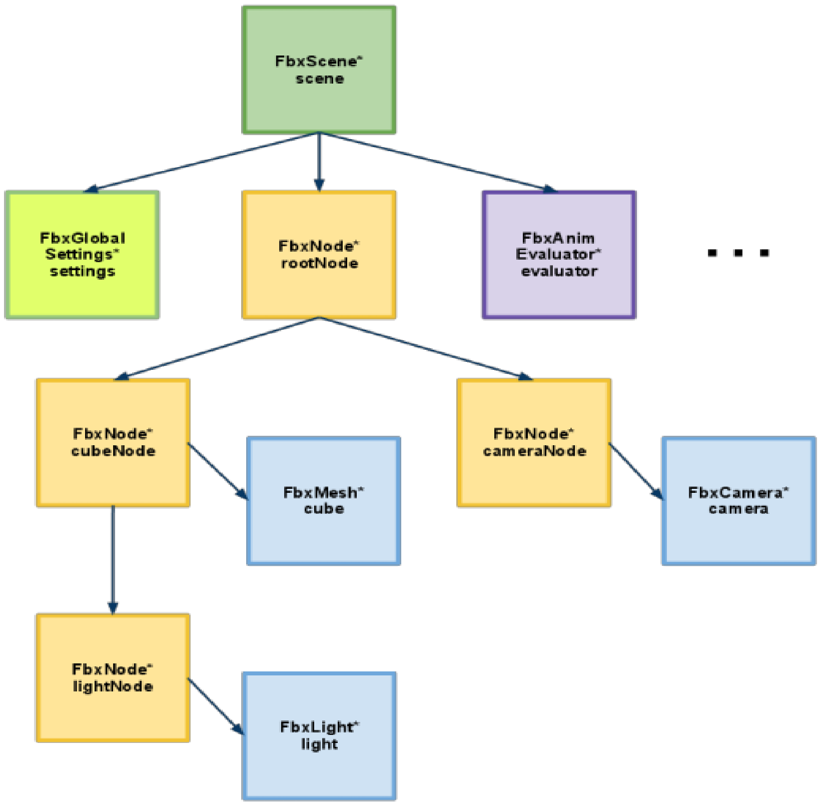
Now, let's check how does it work in the memory.
All object from the **FBX SDK** are managed. This manager is in charge of :
- Allocation
- Deallocation
- Search and access element

For the programmer, it's **easier** to manage **memory**. Upon destruction, the manager **release all the**

objets that he is in charge.
It's possible to set the memory allocator for the manager.
This manager allow us to create other main subsytem, such as the **SceneImporter** or the **Scene**.
Then, we can **traversing** all the hierarchy of nodes, with for example a **recursive function**.

It is very **simple** and **intuitive** to use, here is snapshot of source code from the official FBX SDK documentation :

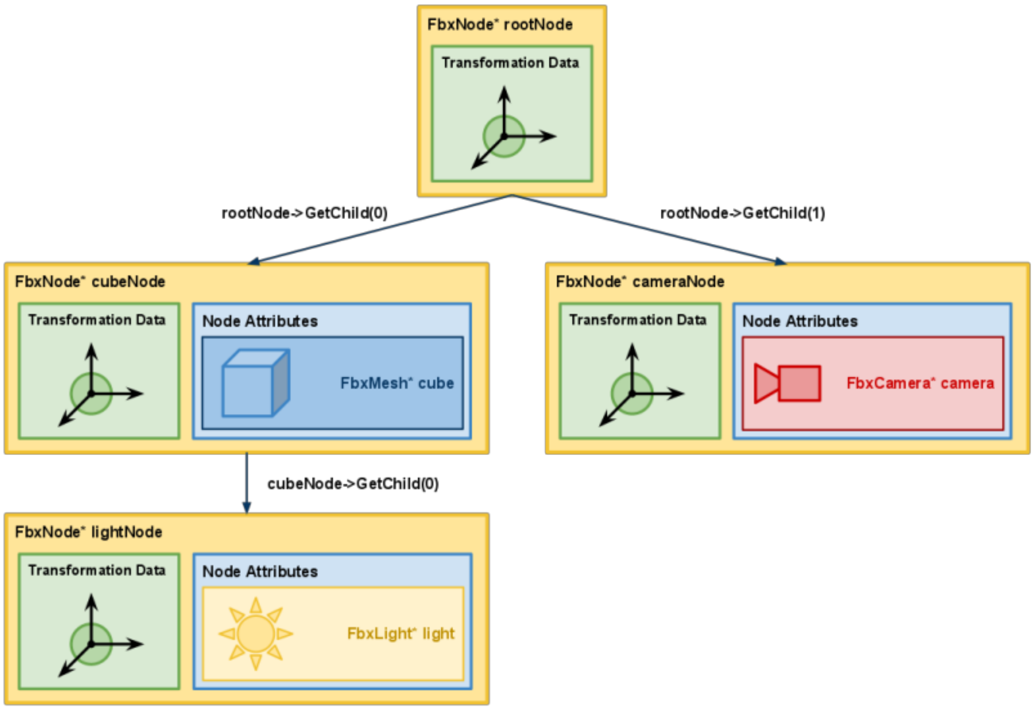
```
// Create the SDK manager.
FbxManager* lSdkManager = FbxManager::Create();

// Create the scene.
FbxScene* lScene = FbxScene::Create(lSdkManager, "Scene Name");

// Get the root node of the scene.
FbxNode* lRootNode = lScene->GetRootNode();

// Create a child node.
FbxNode* lChild = FbxNode::Create(lScene, "child");

// Add the child to the root node.
lRootNode->AddChild(lChild);
```



Appendix 1

this diagram explain how is organized the node hierarchy.

Appendix 2



Here is a more precise diagram of relation between nodes and their attributes

Each node contains his transformations data and their attributes.
Here is a simple snapshot for retrieving a mesh.

```
auto node = scene.GetRootNode(); // get the root node of the scene
auto mesh = node->GetMesh(); // get the mesh attribute from a node

if (nullptr == mesh)
{
    // the node is not a mesh, but could be a light or a camera
}
else
{
    // the node is a mesh, now we can load his vertices, normals, indices or UVs
}
```

Lua lib(lua.hpp, luabridge.h):

Lua?

Lua is a lightweight multi-paradigm programming language designed as ascripting language with "extensible semantics" as a primary goal. Lua is cross-platform since it is written in ISO C. Lua has a relatively simple C API, thus "Lua is especially useful for providing end users with an easy way to program the behavior of a software product without getting too far into its innards
LuaBridge is a lightweight and dependency-free library for mapping data
LuaBridge offers the following features:

- MIT Licensed, no usage restrictions!
- Headers-only: No Makefile, no .cpp files, just one #include!
- Simple, light, and nothing else needed (like Boost).
- No macros, settings, or configuration scripts needed.
- Supports different object lifetime management models.
- Convenient, type-safe access to the Lua stack.
- Automatic function parameter type binding.
- Easy access to Lua objects like tables and functions.
- Written in a clear and easy to debug style.
- Does not require C++11.

How does it work ?

```
local time = 0
local speed = -2

function Update(dt)
        time = time + dt
        if time > 4
        then
                speed = - speed
                time = 0
        end
        Entity.Transform.rotY = Entity.Transform.rotY - speed * dt
end
```

Here is a simple program to Move up a FBX model

vector lib:

Just like arrays, vectors use contiguous storage locations for their elements, which means that their elements can also be accessed using offsets on regular pointers to its elements, and just as efficiently as in arrays. But unlike arrays, their size can change dynamically, with their storage being handled automatically by the container.

sstream lib:

Stream class to operate on strings.
Objects of this class use a string buffer that contains a sequence of characters. This sequence of characters can be accessed directly as a string object, using member str.

stdlib lib:

This header defines several general purpose functions, including dynamic memory management, random number generation, communication with the environment, integer arthmetics, searching, sorting and converting.

memory lib:

This header defines general utilities to manage dynamic memory