Forum: Moteur de jeu GameBlender et alternatives
Topic: hARMful engine
Subject: Re: hARMful engine
PostÃ© par: Bibi09
Contribution le : 6/3/2020 23:53:24

Pour compiler un projet en C/C++ avec des bibliothèques externes, on a besoin de plusieurs choses. Les bibliothèques elles-mêmes (du code compilé en binaire) et aussi ce qu'on appelle la définition des classes, fonctions, données globales, etc (qu'on nomme généralement "header" ou "include").
Ces deux choses, bibliothèques et headers, sont données au compilateur afin qu'il puisse reconnaître des fonctions qui ne sont pas écrites dans le code de notre projet actuel.

Par exemple, tu as une bibliothèque jpeg.dll (si on est sous Windows) pour charger une image JPEG. Dans ton programme, tu utilises la fonction "load_jpeg()" mais tu ne l'as pas écrite dans ton programme. Pourtant, grâce aux "headers" de la bibliothèque "jpeg" que tu lui as donné, il sait que c'est une fonction qui existe même si elle n'est pas dans ton code à toi.

Ensuite, on a une autre étape qui vient après la compilation, c'est l'édition de liens. Ca, c'est pour que l'exécutable sache ensuite, quand tu le lances, que "load_jpeg()" se trouve dans la bibliothèque jpeg.dll (pour faire simple).

Toute cette explication me permet de t'indiquer le rôle de CMake. CMake est un outil multiplateform qui permet, avec un même fichier de configuration, de préparer tout le nécessaire à la compilation d'un projet. C'est CMake qui va faire en sorte que le compilateur reçoive bien toutes les bibliothèques et leurs headers. Il va non seulement vérifier que toutes les bibliothèques (nécessaires ou optionnelles) sont installées mais aussi récupérer toutes les informations à leur sujet. Il va détecter ton OS, les compilateurs installés et tu peux en plus choisir de compiler en 32 ou 64 bits, apporter des options à ton code, etc.
Parmi les fichiers que génère CMake, il y a les solutions Visual Studio sous Windows (si on a Visual Studio). Une solution permet ensuite d'ouvrir le code dans Visual Studio pour faire le développement du projet, le compiler, le déboguer et bien plus encore. C'est un outil extrêmement complet et pas uniquement réservé au C++ (F#, C#, Python...).
Sous Linux, CMake va générer d'autres fichiers pour permettre de compiler facilement.

Là où c'est vraiment super, c'est que le même fichier de configuration peut te permettre de compiler pour différents OS (Linux, Windows, OS X, Android, iOS...).
Exemple de hARMful que tu compiles sous Windows
https://www.youtube.com/watch?v=W5kpHneNB9I et Linux
https://www.youtube.com/watch?v=zECjVr_6qko avec les mêmes fichiers CMake.

Conan est un gestionnaire de paquets dédié aux langages C/C++. Il est en particulier utile pour télécharger des bibliothèques écrites en C/C++.
Conan peut être intégré directement dans CMake. Comme je l'ai dit, CMake vérifie que les bibliothèques dont tu as besoin pour ton projet son bien installées. Or, avec Conan tu peux justement

télécharger les bibliothèques quand celles-ci sont absentes du système (ou si elles sont dans une version trop ancienne). Quand CMake tourne pour générer ses fichiers, il va en profiter - si nécessaire - pour télécharger les dépendances et ensuite les lier au projet. Tu n'as donc rien à faire (car c'est particulièrement chiant), tout est automatisé !

Comme tu développes en Python, tu connais peut-être "pip". Conan c'est comme "pip" mais pour le C/C++. Et Conan peut aussi être installé à l'aide de "pip" (c'est ce que j'ai fait sur mon PC) !
Pour Node.js (JavaScript), il y a npm (Node.js Package Manager). En Rust, le gestionnaire de paquet est cargo. Etc.
Plein de langages ont recours à ce type d'outil tant ça simplifie la vie ! Même Visual Studio propose un tel gestionnaire du nom de NuGet, surtout pour les projets C# (même s'il existe aussi des paquets pour C++, c'est assez marginal et bien moins maintenu que Conan).

Pour ce qui est des assets, c'est toi qui gères. C'est pas un moteur comme Unity où tu as un dossier de projet avec tous tes assets. Je ne fournis que des DLLs qui font une abstraction d'OpenGL au final. Elles ne s'occupent que d'afficher ce que tu souhaites. Il faut juste veiller à ce que, dans le code du programme, le chemin vers les assets soit relatif à l'endroit où se situe le .exe.
Plus tard, j'aimerais bien développer un éditeur graphique pour mon moteur. mais ça ne sera pas avant un long moment...

PS: demain je vais faire un premier patch!
J'ai repéré un gros bug en créant de nouveaux FBX avec Blender qui fait crasher l'application. J'ai déjà la solution à ce bug, je n'ai pas eu le temps de le corriger encore.

PS2: je m'occupe ce week-end de faire une vidéo de présentation du moteur. Le but est de montrer ses (maigres) capacités graphiques ! Pour l'instant, j'ai réussi à mettre en valeur un beau mesh... plein de mordant. Bon, je l'ai récupéré sur BlendSwap car c'est hors de ma portée !