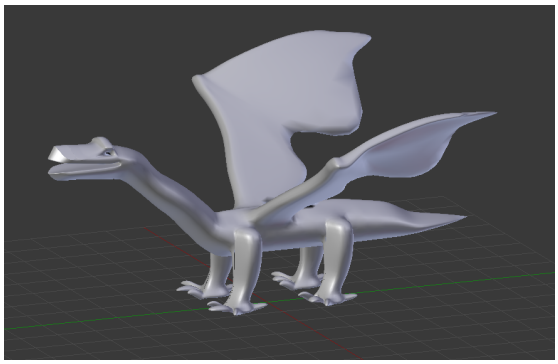
Université de Technologie de Belfort Montbéliard

IN55

Animation d'un personnage 3D avec OpenGL

Florent Jacquet
Romain Thibaud
Antonin Waltz
Superviseur : Fabrice Lauri



Printemps 2016

Table des matières

Table des figures

Chapitre 1

Présentation du projet

Durant ce semestre en IN55, nous avons choisi le projet *Animation d'un personnage 3D* parmi tout ceux proposés. Le personnage que nous avons choisi de modéliser et d'animer est un dragon. Les raisons de ce choix sont multiples.

Tout d'abord, étant tous plus ou moins proche des communautés fan d'heroic-fantasy, le dragon est pour nous une figure emblématique de notre imaginaire. Ce projet nous laisser donc l'opportunité de donner vie à cette créature mythique.

Dans un second temps, c'est l'architecture du squelette qui nous a donné envie. Basiquement, un dragon c'est : une tête, un cou, un buste, des ailes, des pattes et une queue. Cette structure nous a paru suffisament complexe pour que l'on puisse faire des animations intéressantes, mais reste relativement simple n'ayant pas forcément trop d'éléments du squelette qui se suivent. Ainsi, nous avons voulut faire voler, marcher ou s'asseoir notre dragon.

Nous developperons dans ce rapport la structure de notre objet 3D. Nous parlerons aussi des technologies employées pour faire le rendu 3D et de l'architecture de notre programme. Enfin, nous ouvrirons des pistes d'évolution pour ce projet.

Chapitre 2

Modélisation et armature

Nous avons effectué la modélisation du dragon sous Blender. Nous avons choisis ce logiciel pour plusieurs raisons. D'abord son accessibilité, il est plus intuitif qu'une grande partie de ses concurrents (Maya ou 3DsMax) et permet donc d'avoir des résultats convaincants plus rapidement. Ensuite il a l'avantage d'être libre et gratuit, nous n'avons donc pas eu besoin de pirater un logiciel ou d'utiliser des licences étudiantes contraignantes, et nous avons notamment le droit de montrer notre projet. Ceci est relativemment important pour nous, étant donné que nous sommes en dernière année, il est primordial de pouvoir partager nos réalisations dans le cadre de recherche de stage ou d'emploi. Cependant, la suite d'Autodesk (Maya ou 3DsMax) étant majoritairement utilisée dans l'industrie et le divertissement, nous ne sommes pas formé sur ces logiciels en choisissant Blender.

Son armature se découpe en plusieurs parties indépendantes les unes des autres. Il y a :

— La machoire haute (1 os)
— La machoire (1 os)
— Le cou (6 os)
— Le corps allant de la base du coup jusqu'à la queue (8 os)
— Les ailes, indépendantes (13 os par aile)
— Les pattes, indépendantes également (6 os par patte)

Chaque partie est composée de plusieurs os reliés entre eux. Chaque os dispose d'une position et d'une orientation qui dépends de celle de son parent. Sous Blender nous avons utilisé des Inverse Kinematics Bones pour les pattes et les ailes en particulier. En procédant ainsi, un mouvement fluide est obtenu.

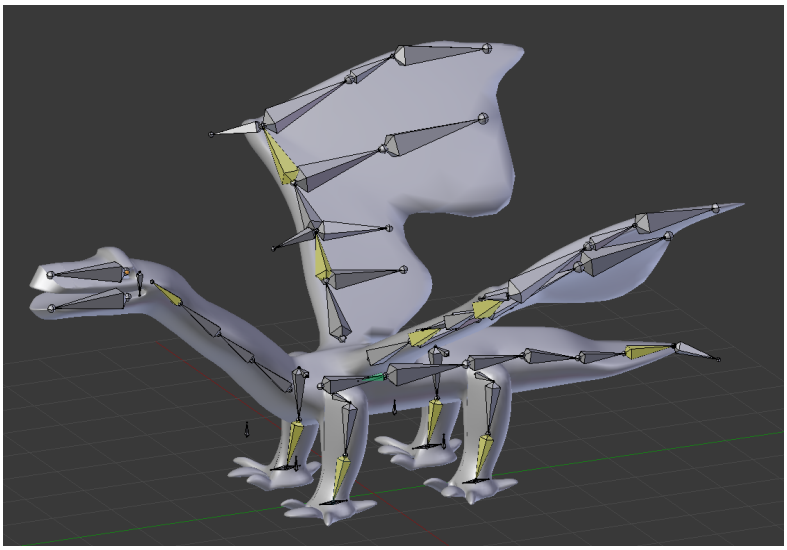


Figure 2.1 – Armature

Chapitre 3

Architecture du projet

3.1 Choix technologiques

De part le sujet du projet, nous devions effectuer le rendu 3D à l'aide d'un programme alliant C++ et OpenGL. Il était conseillé de le faire avec Qt, un framework C++ qui offre diverses bibliothèques et widgets pratique pour mettre en oeuvre des applications C++. Cependant, après de nombreux problèmes dus aux différentes plateformes de développement que nous possédions, nous avons essayer avec la bibliothèque Open GL Utils (qui est très liée à Open GL). Cela nous a permis de faire ce que nous souhaitions et donc nous nous sommes passés de QT.

3.2 La librairie Asset Import

Asset Import est une librairie libre sous licence BSD permettant d'importer dans une structure de données un très grand nombre de fichiers 3D. Elle offre plusieurs avantage, comme le fait d'avoir une API pour du C++ ou du C, ou encore de pouvoir exporter des modèles 3D. Elle est d'ailleurs la base d'une visionneuse 3D disponible sur windows : open3mod (qui est open source). Lors de la modélisation sous Blender, nous n'avions pas à nous préoccupé particulièrement de comment gérer les formes compliquées ou les normales de toutes les faces. Or, lors du parsage du fichier exporté, nous devons nous en préoccuper.

L'utilisation de Asset Import a permis de s'affranchir des opérations bas-niveau de parsage de fichier, ainsi que de la vérification syntaxique du fichier 3D, tout en permettant de faire des tests avec un très grand nombre possible de fichiers comme le Collada, le FBX, le format 3DS, l'OBJ ou même si on le souhaite des format plus exotiques comme le BVH (un format de motion capture) ou le .3d (un format utilisé par le moteur de jeu Unreal).

Ces fichiers supportant différentes fonctionnalités, il n'est toutefois pas toujours possible de visualiser les animations.

3.3 Nos structures de données

Si la librairie assimp permet de charger le fichier en mémoire dans une structure de données, elle ne permet cependant pas de faire de l'animation squelettale directement. Il faut donc soit faire une fonction capable de parcourir la structure rapidement, afin d'extraire les bonnes informations pour ensuite les afficher, mais cette solution est très couteuse en terme de temps de calcul, puisqu'en plus de faire les calculs d'interpolation, il faut aussi gérer le parcours de structure complexe.

Nous avons donc décider de faire nos propres structures de données, plus simples, et donc moins exhaustives, mais suffisantes pour les besoins du projet, tout en nous donnant une maitrise totale sur les options de rendus et d'animations.

Des fonctions de chargement sont donc appellées au lancement, récupérant les informations requises grâce à assimp, pour pouvoir ensuite y accéder rapidement et simplement.

3.3.1 Les structures principales

On trouve généralement dans un fichier une scène contenant plusieurs objets.

— *SceneHandler* : C'est la classe centrale qui contient toutes les informations.
— *Mesh* : Un *Mesh* est une structure qui contient principalement une liste de *vertex*, de *bones* et de *faces*. L'orientation de ces faces est définie par leur normale, comprise également dans le *mesh*.
— *Vertice* : Un *Vertices* contient un *vertex* avec deux tableaux de taille 4. Le premier détaille les *Bones* qui influent sur lui. Le second mesure le poids d'influence.
— *Bones* : Un *Bones* contient un *bone* et la liste des *vertex* sur lesquels il influe.
— *Face* : Une *Face* contient une liste d'indice qui forment une face.
— *Animation* : Une *Animation* contient une liste de *BoneAnim*. Il y a autant de liste que de *Bone*.
— *BoneAnim* : Une *BoneAnim* contient 3 listes qui correspondent aux clés pour la translation, la rotation et la mise à l'échelle.
— *VectInterpolation* et *RotInterpolation* : Ces deux classes s'occupent de l'interpolation des matrices de transformation entre deux *KeyFrame* (moment clé d'une animation)

3.3.2 Caméra libre

La caméra libre dispose de 5 degrès de liberté. Tout d'abord les 3 translations de l'espace sont possibles. Ainsi les flèches du haut et bas nous permette de se déplacer sur l'axe y (axe vertical) pendant que celle de droite et gauche nous translate selon x (axe horizontale) enfin la molette de la souris contrôle le mouvement suivant z (la profondeur). Ensuite 2 rotations sont disponibles. Un clique gauche souris maintenue suivi d'un déplacement permet de tourner autour de l'axe vertical tandis que la même manipulation avec un clique droit permet une rotation autour de l'axe horizontal. La rotation autour de l'axe de profondeur n'ayant que peu d'intérêt, elle n'a pas été implémenté.

Le fonctionnement est relativement simple, il y a un *eventListenner* qui écoute les commandes claviers, un autre pour celle de la souris et un dernier pour les mouvements de la souris. Si un de ces trois éléments reçoit un stimuli, il met à jour la position et ou l'orientation de la caméra, et le rendu de cette dernière sera effectué au prochain rafraîchissement de l'affichage.

Chapitre 4

Diagramme de classe

4.1 Mesh Structure

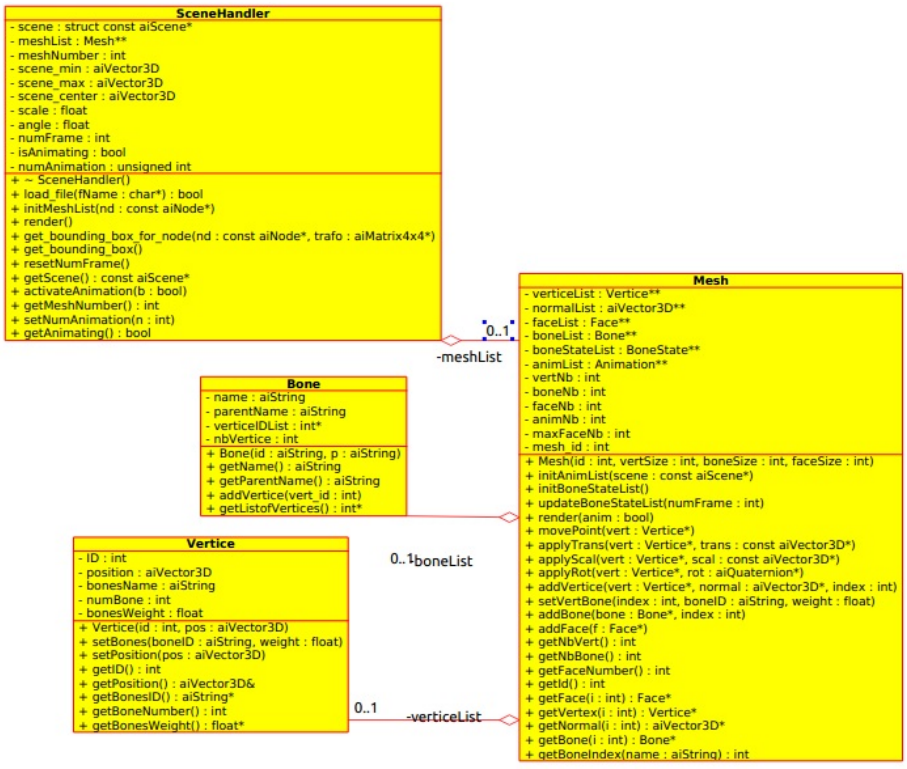


Figure 4.1 – Diagramme de classe pour la structure d'un Mesh

La structure de Mesh est ce qui nous permet d'effectuer le rendu de notre objet. Elle est composée d'une liste de *bones* et une de *vertices*. On peut ainsi savoir qui est lié à qui. De plus, dans chaque *vertice* on stocke aussi l'importance des *bones* sur lui dans le tableau de *Weights*. Le nombre de *bones* est

volontairement limité à 4 pour alléger un peu les calculs, et d'après l'avis de plusieurs utilisateurs de la biliothèque, ce nombre est suffisant.

Il y a aussi une liste de face qui permet de savoir quelles vertices forment quel polygone. On stocke avec les normales à chaque face, ce qui est très utile pour les rendus d'illumination par exemple.

La fonction qui procède au rendu est la fonction *render*. Dans le *SceneHandler* on appelle cette fonction pour chaque *Mesh* qui va parcourir la liste des faces afin d'en effectuer le rendu. C'est d'ailleurs ici, que l'on effectue les transformations apportées aux vertices lors de l'animation afin d'en faire le rendu.

Toutes nos listes sont initialisées grâce à la méthode *initMeshList* de *SceneHandler* et ce avant le premier rendu ce qui nous permet d'effectuer certains calculs en amont. D'où l'intérêt de se servir de nos propres structures.

4.2 Animation Structure

Figure 4.2 – Diagramme de classe pour la structure d'une animation

La classe *Animation* est celle qui nous permet de stocker toutes les animations du mesh. Elle contient une liste de *BoneAnim*, c'est à dire toutes les *KeyFrames* associées à un *bone*. Ces *KeyFrames* sont stockées sous trois variables : translation, mise à l'échelle et rotation. Tout est initialisé dans la méthode *initAnimList*.

A chaque frame, on actualise la liste de *BoneState*, des matrices $4x4$ représentant la transformation associée à chaque bone à la frame donnée. Cette opération est effectuée dans la méthode *updateBoneStateList*. Elles seront ensuite utilisées dans la méthode *render* pour faire bouger les vertices.

Il existe aussi trois fonctions d'interpolation qui gèrent l'interpolation pour les translations, les mises à l'échelle et les rotations. Elles sont utiles pour calculer le *BoneState*, puisque qu'on interpole chaque transformation avant de créer la matrice commune.

Chapitre 5

Bilan

5.1 Guide d'utilisation

Voici les touches à utiliser pour manipuler le programme :

1,2,3 : Sélectionne et lance une animation.

$\leftarrow, \uparrow, \downarrow, \rightarrow$ **:** Translate la caméra sur les axes X et Y

Souris : Maintenir un clic et déplacer la souris déplacera la caméra libre sur l'axe en question.

Molette souris : Avancer ou reculer la molette déplacera la caméra de façon à zoomer/dézoomer.

5.2 Difficultés rencontrées

Durant ce projet, nous avons été confronté à plusieurs difficultés :

— La prise en main des Inverse Kinematics pour avoir un rendu acceptable des animations dans Blender a été laborieuse.

— Il nous a fallut un temps d'adaptation pour prendre en main et comprendre comment utiliser la librairie Assimp.

— Comprendre comment parcourir de grandes quantités de données à travers des structures complexes pleines de références croisées a également été un frein à l'avancé du projet.

— De manière générale, la gestion de la mémoire en C++.

5.3 Améliorations possibles

En l'état il existe plusieurs types d'améliorations qui pourraient être apportées au projet. Tout d'abord, on pourrait texturer notre modèle. Actuellement on reconnait la forme d'un dragon mais pas l'aspect. En rajoutant une texture et des *normalmap* et autres *bumpmap*, on pourrait donner un aspect reptilien à la surface du dragon, affiner certains aspects et rajouter des formes sans pour autant qu'il y ait plus de polygones. Cela impliquerait de mettre en oeuvre des Shaders pour effectuer les calculs d'illuminations par exemple.

Les Shaders pourraient aussi nous aider à affiner l'animation en allégeant la charge de calcul du CPU en se servant du GPU pour appliquer les transformations. En effet, le parcours de strcuture suivit du calcul des tranformations allourdit la charge de calcul, et en déléguant au GPU on pourrait alors charger des scènes plus lourdes et que cela reste fluide.

La caméra libre pourrait aussi être améliorée, son utilisation est actuellement un peu rude, surtout au niveau des rotations. Il s'agirait de modifier les valeurs de modification de la position et de la rotation. Par exemple, pour la rotation, on base le changement sur l'amplitude du mouvement de la souris, celui-ci mériterait d'être traiter pour que se soit plus agréable et on pourrait rajouter une zone de non-action autour du curseur, afin qu'un petit tremblement ne bouge pas la caméra.

5.4 Conclusion

Pour conclure, ce projet a été l'occasion de découvrir la programmation graphique avec OpenGL et l'utilisation de librairies gravitant autour de la 3D. Travailler sur un dragon et le voir s'animer au fil du temps a rendu le projet particulièrement ludique. Nous avons pu appréhender le processus de modélisation et d'animation d'un personnage pour arriver à la représentation du rendu dans une scène. Nous avons pu constater comment appliquer les concepts vus en cours. Explorer les différentes possibilités offertes par des librairies libres nous a aussi forcer à aller chercher ce qu'il est possible de faire et comment le réaliser avec ces outils mis à notre disposition.