Animation d'un personnage en 3D

Sibel Bedir, Floriane Ennaji, Lucille Gomez

IN55, UTBM - Printemps 2016

Table des matières

Introduction

Ce projet s'inscrit dans le cadre de l'UV de IN55 : *Synthèse d'images*. Le but de ce projet est d'animer un personnage en 3D avec Qt en utilisant Opengl. Pour ce faire, il faut d'abord le modéliser sous un modeleur 3D, puis effectuer le rendu de plusieurs animations dans une scène. Le rendu se fait en temps réel et une caméra libre et mobile est également implémentée.

Dans une première partie, nous allons présenter l'architecture globale du projet. Suite à cela, nous aborderons la création du personnage 3D sous Blender, et nous expliquerons comment nous avons animé et fait le rendu de ce personnage sous QT, avec une caméra notamment. Pour finir, nous évoquerons les difficultés rencontrées pendant ce projet et les éventuelles solutions apportées.

Chapitre 1

Architecture du projet

1.1 Description

Les objectifs principaux du projet d'Animation de personnages 3D sont les suivants :
— modéliser un personnage sous un modeleur 3D
— effectuer le rendu de plusieurs animations dans une scène
Ce sujet a des objectifs communs à tous les projets proposés dans le cadre de l'UV IN55 :
— rendu temps-réel
— implémentation d'une caméra libre
Le projet a été découpé en trois parties distinctes : la modélisation du personnage, son animation, et l'implémentation de la caméra.

1.2 Diagramme de classe

Le diagramme de classe complet se situe en annexe pour des raisons de visibilité. Le diagramme ci-dessous a été simplifié : en enlevant les attributs et les fonctions. Il permet de visualiser rapidement les différents liens entre les classes présentes dans le projet.

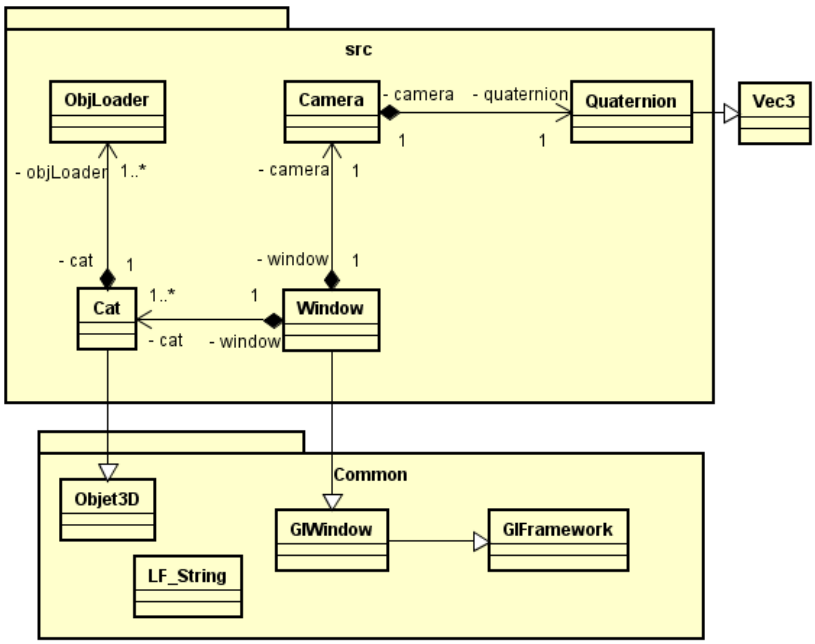

Figure 1.1 – Diagramme de classe simplifié

La partie modélisation n'est pas représentée dans le diagramme de classe ci-dessus. Cependant, elle est reliée à la partie animation via la classe *ObjLoader*. Cette classe permet de lire et de stocker les données d'un fichier .obj généré à partir du modèle 3D.

Afin de représenter le personnage 3D implémenté (un chat) dans une fenêtre et de l'animer, nous avons créé la classe *Cat*. Cette classe hérite de la classe *Object3D*. Un chat contient un ensemble de tableaux récupérés par la classe ObjLoader, comportant ses coordonnées dans l'espace 3D, ses coordonnées de texture, et ses coordonnées de normales.

Le chat va être animé grâce à plusieurs fonctions qui opèreront sur ses différents tableaux de données.

Le rendu se fait dans une fenêtre sous Qt, avec différentes fonctionnalités. La classe *Window* hérite de la classe *GlWindow* et permet de visualiser le chat. Grâce à la classe *Camera* qui utilise la classe *Quaternion*, il est possible de changer le point de vue sur la scène en zoomant, se déplaçant etc.

Chapitre 2

Création du personnage 3D

Nous avons décidé d'utiliser le logiciel Blender pour modéliser notre personnage 3D. Afin de faciliter sa prise en main, nous avons suivi un tutoriel très complet [1] sur Youtube. On peut y suivre la création d'un petit bonhomme, depuis la modélisation de son mesh (maillage) jusque son animation tout en passant par l'application de textures, etc. Comme nous avions décidé de créer un chat, il a fallu adapter différentes étapes de la modélisation.

2.1 Création du mesh

Un modèle 3D existe avant tout sous la forme d'un mesh. Il s'agit d'un ensemble de sommets (vertices), d'arêtes et de faces définissant la forme d'un objet.Nous avons commencé la création du mesh en partant d'un cube. Nous avons modélisé en premier la tête du chat. C'est la partie la plus compliquée et celle qui a pris le plus de temps.

Nous avons utilisé trois images de fond, représentant un même chat depuis trois points de vue différents : vue de face, vue de côté gauche et vue de dessus. Grâce à cela, on garde des proportions assez réalistes lorsque l'on modélise. En 3D, il est très facile d'avoir des personnages d'allure correcte dans une vue, mais totalement déformés dans une autre.

Une fois la tête créée, nous nous sommes intéressées au corps, puis aux pattes, et pour finir à la queue du chat. A la fin de cette étape, nous avions ainsi un chat en plusieurs morceaux. Suite à un changement de décision quant au type d'animation que nous allions implémenter, ces différents morceaux ont été joins afin de ne former qu'un seul et unique mesh. Par la suite, nous avons appliqué un modificateur sur le mesh (triangulate) pour résoudre des problèmes d'animation.

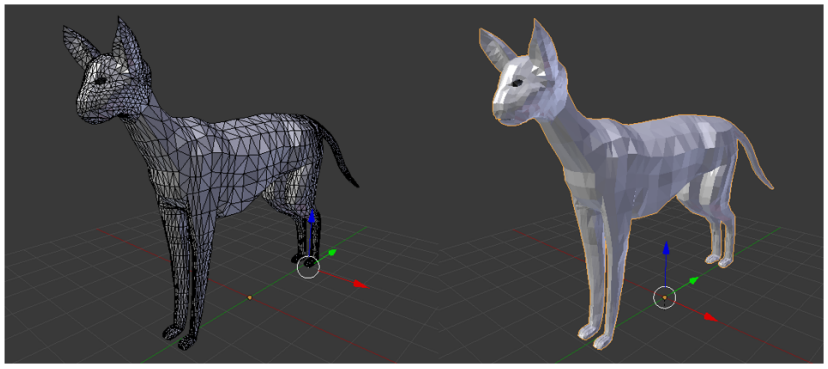

Figure 2.1 – Mesh final du chat dans Blender

2.2 Matériau et textures

Nous avons décidé d'appliquer une texture simple sur notre chat afin de le rendre plus réaliste. Nous avons tout d'abord projeté les différentes paries du corps du chat sur une image 2D (UV unwrapping). Ensuite, nous lui avons appliqué un matériau de test, une UV grid, afin de vérifier et modifier la répartition de la future texture sur le matériau : c'est le checker pattern. En effet, selon le relief de l'objet, la texture peut être étirée ou compressée, donnant un rendu non réaliste.

Une fois que le matériau est créé, nous pouvons lui appliquer une texture. Cette dernière a été créée en utilisant des couleurs extraites de textures de poils de chats, pour un effet plus lisse. En effet, le mesh du chat contient de nombreuses aspérités donc utiliser une image préexistante comme texture donne un rendu étrange.

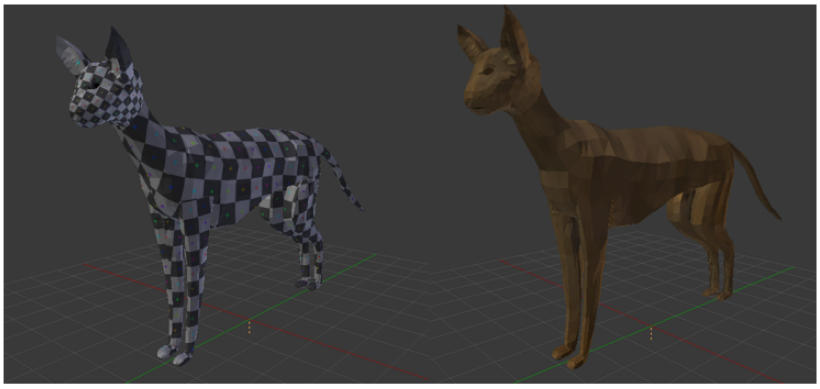

Figure 2.2 – Checker pattern et rendu final de la texture dans Blender

2.3 Création de l'armature

Nous avons décidé d'animer notre chat en l'interpolant entre différentes positions. Afin de simplifier la création de différentes positions, nous avons lié une armature (squelette) au

mesh du chat. Une armature est constituée de bones (os) qui vont être attachés via un degré de parentée avec le mesh. Ces bones auront une influence plus ou moins forte sur les membres au niveau desquels ils se situent. Nous pouvons ainsi déformer le mesh sans avoir à le modifier directement grâce à cette armature.

Même si l'armature n'est pas utilisée par la suite, la créer a permis un gain de temps considérable et une animation plus réaliste entre les diverses positions générées. A partir de la position initiale où le chat se tient simplement debout sur ses quatre pattes, nous avons créé des positions pour la marche, la course ainsi que le saut. Afin d'utiliser ces modèles Blender, nous les avons exporté en fichiers de type .obj.

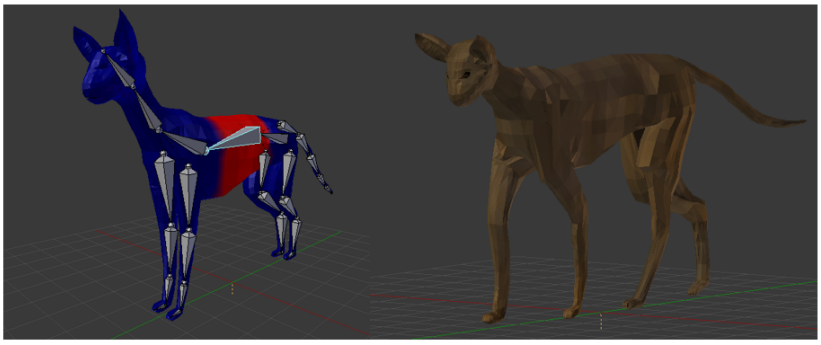

Figure 2.3 – Armature, influence d'un bone, et position de marche

Chapitre 3

Animation

3.1 Exportation du modèle 3D

Les modèles 3D générés sous Blender ont été exportés sous le format .obj afin de les utiliser sur Qt. Pour ce faire, nous avons implémenté un parseur de fichiers .obj.

Le .obj généré contient 4 types de données :
— Un tableau de vertices : il s'agit des coordonnées de chaque vertex du mesh, de la forme $[x, y, z]$
— Un tableau de coordonnées de texture : il s'agit des coordonnées de texture de chaque vertex, de la forme $[x, y]$
— Un tableau de normales : il s'agit d'un tableau de normales de la forme $[x, y, z]$
— Un tableau de faces : chaque face est définie par un tableau de la forme $[v1, vt1, vn1]$ $[v2, vt2, vn2][v3, vt3, vn3]$, reliant ainsi trois vertices via leurs coordonnées dans l'espace, leurs coordonnées de texture et leurs normales

3.2 Technique d'animation

Les animations que nous avons implémentées sont la marche, la course et le saut. Pour les visualiser, il suffit de taper sur la touche **W** pour la marche, **R** pour la course et **J** pour le saut. Il suffit de taper sur la même touche ou une des deux autres touches d'animation pour arrêter l'animation en cours.

L'animation du chat se fait par interpolation des coordonnées de vertices. En effet, nous avons décidé d'utiliser la méthode de l'animation par *keyframes*. Il s'agit d'importer le même personnage dans différentes positions - chaque position étant une étape d'un mouvement, et de réaliser une interpolation entre les coordonnées de ses vertices au départ et à l'arrivée de chaque étape du mouvement.

Si l'on veut rendre un mouvement fluide, il faut le décomposer en un juste nombre de positions. Chez les animaux, les différentes cadences sont déterminées en nombre de temps. La marche a souvent entre deux et quatre temps, la course également. Nous nous sommes basées sur ce principe de temps pour choisir les positions des mouvements. Ainsi, dans notre projet, une animation comme la marche ne contient que deux positions, tandis que le saut en

contient sept.

L'interpolation en tant que telle se fait pour chaque vertice entre les deux vecteurs de coordonnées. Nous utilisons la méthode Linear intERPolation, LERP. Le calcul de la position d'un vertex à un état intermédiaire est donné par $position = origine + completion * (destination - origine)$, où *origine* et *destination* sont les coordonnées du vertex à l'origine et à l'arrivée de l'étape du mouvement. *completion* correspond au pourcentage de l'animation, sous forme de `float` compris entre 0 et 1.

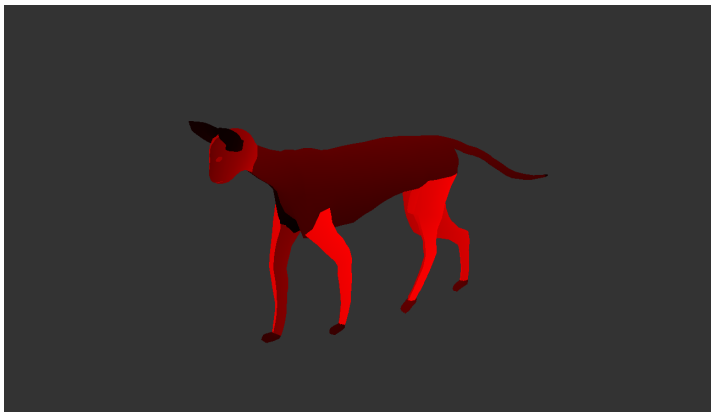

Figure 3.1 – Interpolation entre deux étapes de la marche

Afin de donner au chat une allure réaliste, la durée d'une étape dépend du nombre total d'étapes dans un mouvement. La *completion* envoyée à l'interpolation est en fait une fraction du nombre total de périodes intermédiares entre le début et la fin d'une étape d'un mouvement. Concrètement, la *completion* d'une étape la marche passe de 0 à 1 en 36 appels, tandis que celles de la course et du saut le font en 12 appels. Les étapes de début ou de fin d'animation (depuis ou vers la position immobile du chat) ont moitié moins de périodes intermédiares que celles du mouvement concerné.

3.3 Caméra

La caméra utilisée dans ce projet est une caméra libre et mobile. En effet, la caméra elle-même et le regard peuvent se déplacer. Pour l'importer dans la fenêtre, nous nous servons de la fonction de prototype `void lookAt(GLfloat eyeX, GLfloat eyeY, GLfloat eyeZ, GLfloat targetX, GLfloat targetY, GLfloat targetZ)` définie dans la classe `GlFramework` donnée. Le paramètre `eye` correspond à la position de la caméra et `target` correspond à son orientation.

La caméra change de position lorsque l'utilisateur utilise les touches du clavier ou la molette de la souris. Concrètement, les touches **flèche vers le haut** et **flèche vers le bas** vont déplacer la caméra sur l'axe Y. Les touches **flèche vers la droite** et **flèche vers la gauche** vont déplacer la caméra sur l'axe X. Enfin, les touches + et **-**, ainsi qu'un *scroll* sur la molette vers le haut ou vers le bas vont déplacer la caméra sur l'axe Z. Les déplacements

de la position de la caméra sont des transitions de facteur 1 sur l'axe.

Des positions prédéfinies de la caméra sont implémentées. Elles ne changent que la position de celle-ci, en gardant toujours l'orientation sur le chat. Pour les utiliser, il suffit d'utiliser les touches **0** (position de départ), **1**, **2**, **3**, **4** ou **5**.

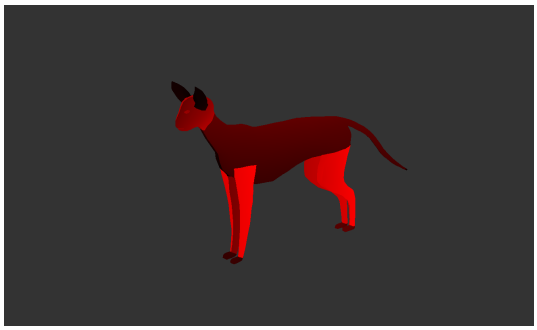

Figure 3.2 – Position prédéfinie 1

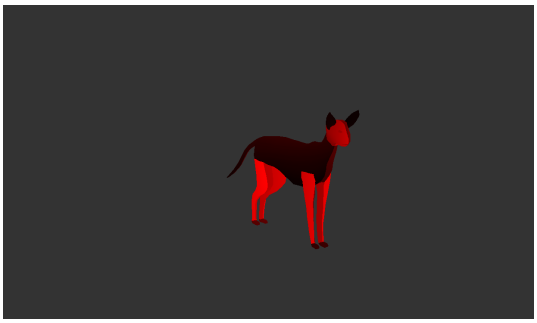

Figure 3.3 – Position prédéfinie 2

La caméra change d'orientation lorsque l'utilisateur bouge sa souris. Pour activer le mode changement d'orientation, il doit d'abord cliquer. Ensuite, la caméra suivra les mouvements de la souris et déplacera l'orientation de la caméra sur les axes X et Y. On ne déplace pas l'orientation sur l'axe Z car la fenêtre sur laquelle l'utilisateur déplace sa souris n'est qu'en deux dimensions. Concrètement, on capte la position de la souris sur la fenêtre et on effectue une translation de facteur $\frac{position - centre}{sensibilite}$ pour les axes X et Y. Le centre de l'axe est calculé à partir de la largeur ou hauteur maximale de la fenêtre car le repère de la fenêtre est différent de celui de la caméra. Pour plus de facilité d'utilisation, on utilise un facteur de sensibilité de la souris et un seuil maximal de rotation de l'orientation. Pour désactiver le mode d'orientation, l'utilsateur doit cliquer à nouveau.

Chapitre 4

Difficultés rencontrées

4.1 Animation et modèle

Un temps non négligeable a été consacré à la modélisation 3D. En effet, en plus de devoir apprendre à utiliser Blender, nous avons changé plusieurs fois le type d'animation que nous allions utiliser et cela a influencé des choix de modélisation.

Tout d'abord, nous nous étions basées sur une animation semblable à celle de l'humanoïde créé en TP : chaque partie du corps est affichée en fonction de la position de la partie dessinée précédemment. Ensuite, une ou plusieurs rotations et translations sont appliquées directement sur ces différentes parties du corps afin de créer le mouvement. Pour de créer une animation fluide et réaliste, chaque partie du corps du chat a été séparée au niveau des articulations. Par exemple, la patte avant gauche a été séparée en 4 parties. Cependant avec cette méthode, animer le chat devenait plutôt complexe et compliquée à mettre en place.

Nous nous sommes alors intéressées à l'animation utilisant un squelette relié aux différentes parties du corps. Mais nous nous sommes rendu compte que les fichiers .obj générés par blender n'incluaient pas le squelette. Il n'était donc pas possible d'agir directement sur les bones dans Qt pour animer le chat.

Nous avons donc finalement décidé de créer plusieurs modèles du chat sous différentes positions, afin de l'animer en interpolant entre deux modèles différents. Cette solution impliquait que le chat soit constitué d'une seule pièce afin de réduire le nombre de calcul et simplifier le rendu. Il a donc fallu modifier le mesh pour supprimer tous les vertex qui devenaient inutiles.

Une fois l'animation faite, nous nous sommes aperçues qu'il y avait des problèmes au niveau du mesh du chat lors des différents mouvements. En effet, certaines faces du mesh effectuaient une rotation, au niveau du dos, de la queue, et des connections entre les pattes et le corps. Afin de résoudre ce problème, le mesh a été modifié afin de supprimer toutes les faces de plus de 4 côtés, puis nous lui avons appliqué un modificateur : triangulate. Ce modificateur permet de changer toute face en un ensemble de triangles. Après avoir appliqué ce modificateur, il a fallu tout recommencer : appliquer un matériau, créer et appliquer une texture, créer une armature et la lier au mesh. Grâce à ce changement, le chat apparaît beaucoup plus lisse dans Qt, et le problème de rotation des faces du mesh est résolu.

4.2 Technique d'animation

Comme nous avions déjà implémenté une classe `Quaternion` pour la caméra, nous avons d'abord pensé à interpoler les vertices des positions avec l'aide de ceux-ci. Dans le TP sur la caméra, le diagramme de la classe `Quaternion` indique que celle-ci a une fonction `SLERP`. Nous nous sommes donc renseignées sur l'utilisation de l'interpolation linéaire sphérique.

```
Quaternion Quaternion::slerp(const Quaternion& q1, const Quaternion& q2, float t)
{
    Quaternion quat1 = q1;
    Quaternion quat2 = q2;
    quat1.normalize();
    quat2.normalize();
    Quaternion quat_prod = quat1.operator *(quat2);
    float teta = acos(quat_prod.m_W);
    Quaternion quat;

    quat=(quat1.operator*((sin(teta)*(1-t))/sin(teta))).operator+(quat2.operator*((sin(teta)*t)/sin(teta)));

    return quat;
}
```

Figure 4.1 – SLERP entre deux quaternions

Nous avons donné en paramètres les quaternions instanciés avec les points de départ et d'origine, et *completion*. Cependant, le rendu obtenu n'était pas du tout celui attendu : il y avait bien une animation, mais il semble que toutes les transitions se faisaient à partir du centre. La forme générale de l'objet était une sphère et non un chat. Ce rendu peut être expliqué par le fait que, comme on normalise les quaternions, les coordonnées finales étaient nettement plus petites que les coodonnées de départ. En effet, les coordonnées de vertices extraites du .obj ne sont pas normalisées. Faute de temps et après de multiples essais, nous avons mis cela de côté et avons implémenté LERP. Cette méthode a donné un rendu très satisfaisant.

4.3 Importation de la texture

Nous avons décidé de créer la texture du personnage sur Blender, puis de l'importer dans Qt afin de l'appliquer.

Pour cela, nous avions besoin de lier les vecteurs de texture du fichier .obj à l'image de la texture. Dans un premier temps, nous avons exporté la texture en .tga puis l'avons convertie en .dds, ce qui permet de ne pas perdre en qualité et de gagner en performance comparé à l'utilisation d'un fichier .png par exemple.

Une fois le fichier .dds importé en tant que texture, nous avons modifié les shaders afin qu'ils prennent comme paramètres un `Vec2` correspondant au vecteur texture du .obj en entrée pour le vertex shader, et un sampler2D correspondant à la texture créée à partir du .dds en uniform pour le fragment shader. Après l'affectation des valeurs, la texture n'était toujours pas visible. Nous n'avons pas réussi a résoudre ce problème qui peut venir de l'affectation en elle-même ou de la convertion du fichier .tga en .dds.

Conclusion

Réaliser ce projet nous a permis de comprendre plus en profondeur les différentes techniques d'animation d'un personnage et de nous familiariser avec le développement Opengl. De plus, même si aucune de nous n'avait déjà modélisé un objet en 3D, nous avons choisi de le faire nous-même et cela nous a apporté énormément.

Bien sûr, nous avons eu des difficultés dans toutes les étapes de la conception du projet. À l'exception du chargement de la texture sous Qt, nous avons quand même su trouver de bons moyens pour afficher notre personnage et l'animer.

Le projet comporte encore des points qui peuvent être améliorés. Tout d'abord, nous pourrions créer une surface correspondant au sol, sur lequel le chat pourrait se déplacer. Cela rendrait l'utilisation de la caméra mobile plus intéressante. Enfin, nous pourrions donner à l'utilisateur la capacité de faire le mouvement en restant appuyé sur la touche, à la manière d'un jeu video.

Bibliographie

[1] Darrin Lile, Blender Character Modeling, Modifié le 7 mai 2016,
Disponible sur https ://www.youtube.com/playlist ?list=PLyelx0TsmSpcnM61Z2XLrs1sInsQvdfl0
Consulté le 01/06/2016

Annexes

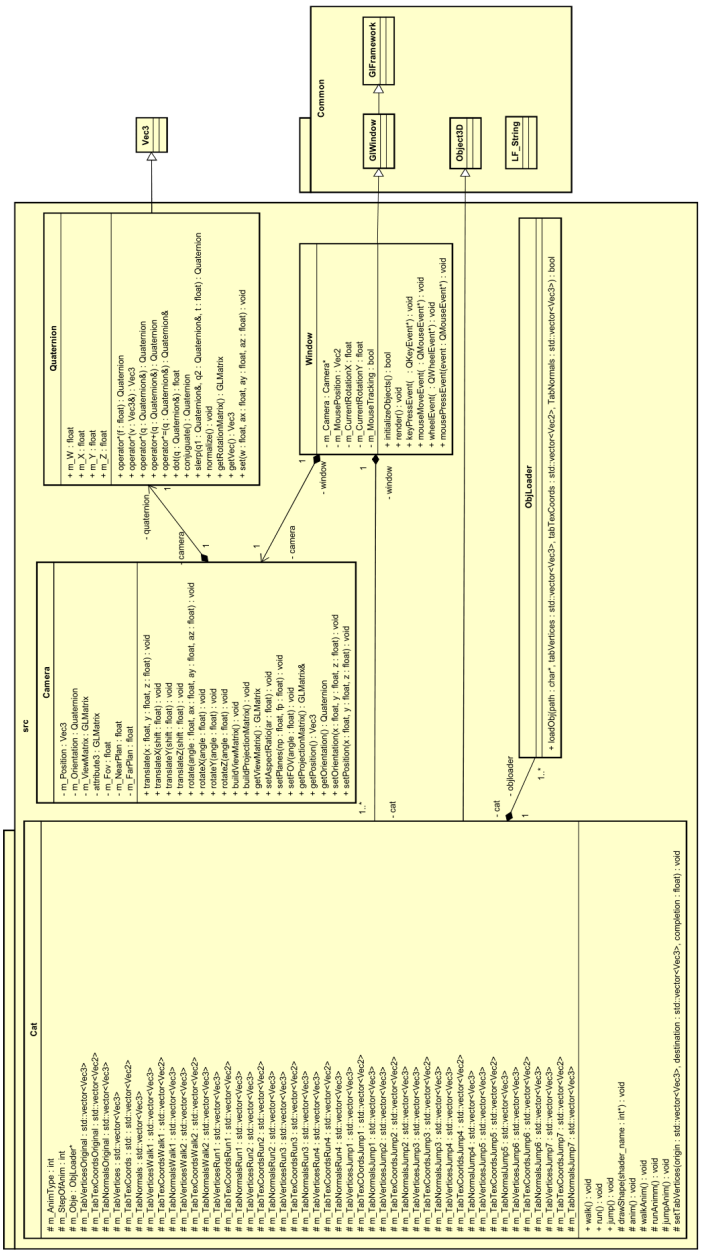

Figure 4.2 – Diagramme de classe complet