Université de Versailles - Saint-Quentin
Projet de troisième année de licence Informatique

Pipeline de rendu graphique 3D

Compte rendu

GROUPE DE PROJET

Amine **ATEK**
Guillaume **BIGAND**
Arthur **HERBLOT**
François-Xavier **MORDANT**
River **MOREIRA-FERREIRA**
Louis-Wilhelm **RABAN-SCHÜRMANN**
Fiona **SANTORO**
Vincent **XAVIER**

ENSEIGNANTE
Leila **KLOUL**

25 mai 2021

Table des matières

1 » Introduction

1.1 But du projet

Notre projet consiste en l'implémentation **d'un moteur de rendu 3D**. L'objectif est donc de permettre l'affichage d'objets tri-dimensionnels sur un écran bi-dimensionnel.
Pour cela, les objets 3D doivent passer par plusieurs étapes afin de permettre leur représentation fidèle et réaliste. Ce sont ces étapes qui sont implémentées dans notre projet, en plus de **deux exemples d'utilisation sous forme d'application** de cette pipeline : une application de conception d'un aménagement intérieur et une simulation d'un système stellaire.

La structure d'une telle Pipeline graphique 3D [1] peut se diviser en deux parties principales : **la géométrie** et **la rastérisation** (ou matricialisation).

1.2 Historique et cas d'utilisations

La "Géométrie", ou la **mesure de notre environnement** s'est développée sur le plan tridimensionnel par l'intérêt qu'avaient les scientifiques de la Grèce antique pour l'astronomie. Les premiers outils développés, comme la *machine d'Anticythère*, étaient basés sur les mêmes principes qui nous permettent aujourd'hui de faire des calculs tridimensionnels complexes.
Dans les années 1960, les besoins de l'industrie ont permis la création des premiers systèmes de *CAO* [6], utilisant de la **modélisation géométrique** pour créer virtuellement un objet, le visualiser et le tester pour l'améliorer avant sa fabrication.

La **visualisation** et la **simulation** qu'ont permis ces premiers moteurs graphiques 3D ont contribué à l'avancée de la recherche, **de la biologie et la médecine à l'archéologie,** par ses possibilités de prédiction pour mettre en image des hypothèses et explorer en détail des contenus ; et ont facilité **les projets architecturaux et la conception industrielle.** Aujourd'hui, c'est la **dimension ludique et artistique** qui s'est approprié la technologie avec notamment les studios jeux vidéo et le cinéma.

1.3 Intérêt actuel du projet

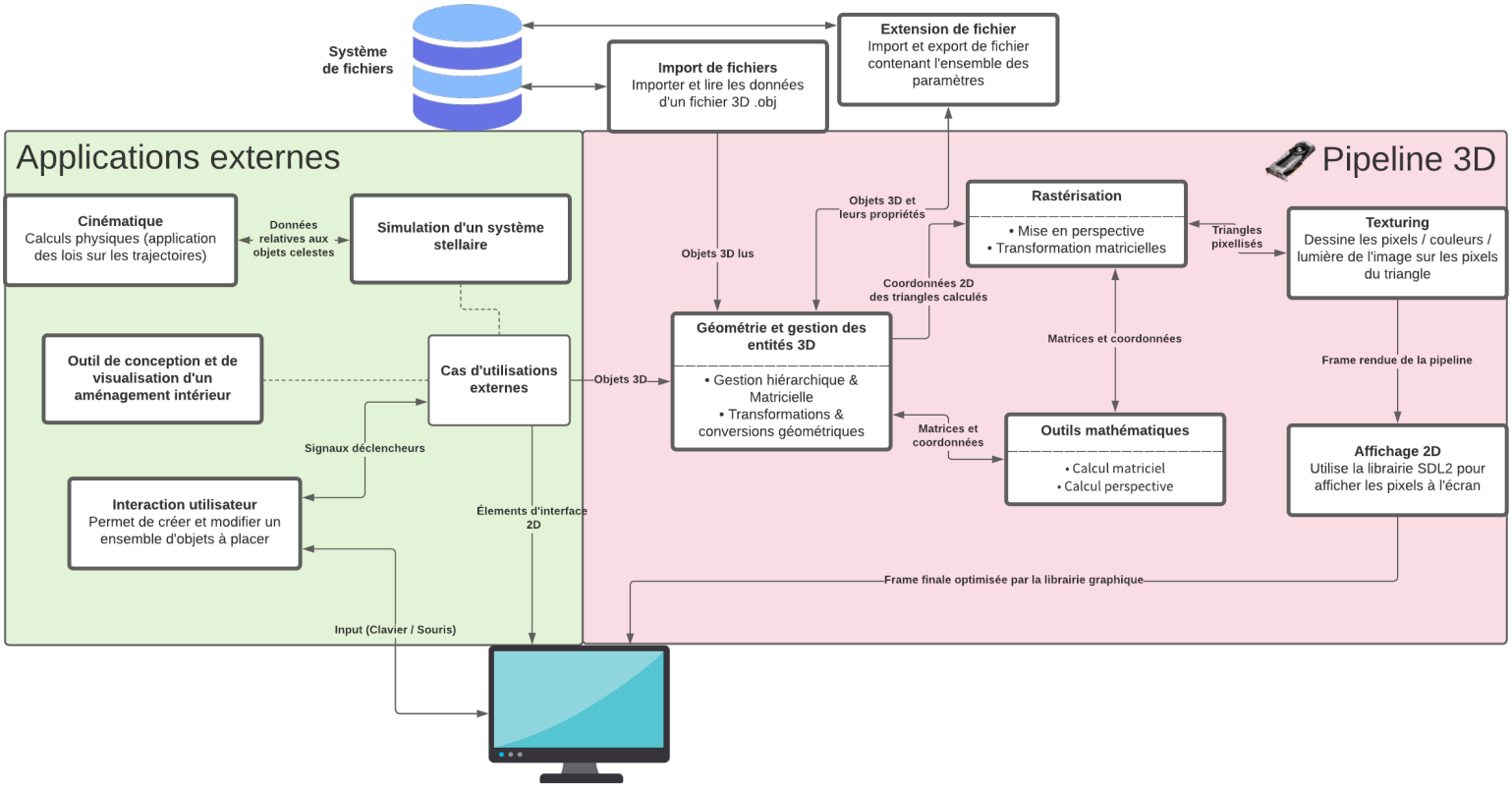
Le programme à l'issue du projet est une librairie graphique 3D optimisée et simple d'utilisation destinée aux développeurs d'applications de tous les domaines d'utilisation listés en introduction, leur permettant de **simuler, visualiser et modéliser des éléments 3D et de se déplacer dans un environnement tridimensionnel.**

De nombreuses collections de bibliothèques 2D existent en `open-source` ; ce n'est cependant pas le cas pour les moteurs effectuant des **conversions matricielles 3D.** Ceci en plus de notre passion antérieure pour le domaine justifie notre volonté de développer un tel projet.

2 » Architecture du projet

2.1 Organigramme

La **première partie de ce projet** a résulté en le découpage de notre projet en **cinq modules propres** au moteur de rendu 3D en lui même, un module **d'import/export de fichier** prenant en compte le type `Wavefront .obj` [7], ainsi qu'un module **d'interaction utilisateur** permettant de faire le lien avec les **deux modules d'application externes.**



Les 3 modules principaux constituent le cœur des **étapes de traitement de données 3D** conduisant à leur transformation pour leur affichage à l'écran. Les liens notables entre ces modules sont donc des données et caractéristiques **d'objets 3D décomposés et transformés** par chacune des étapes - lesquelles sont décrites dans la section suivante - en triangles de maillage [2], puis en triangles augmentés d'une mise en perspective, et enfin en une matrice de pixels.

Le nom de **Pipeline** [1] indique que ce procédé est **répété en continu** pour générer à chaque `tic` une nouvelle `frame` comprenant **l'ensemble des objets 3D traités** selon leurs position et rotation variables.
Les objets 3D à traiter peuvent être chargés depuis un fichier externe (.obj, fichier d'application ou fichier de Pipeline), ou alors depuis les **appels d'applications externes** que nous avons développé sur le moteur de rendu via l'implémentation préalable d'un **module interaction utilisateur** permettant la **création d'interface graphique et la liaison de signaux** sur la librairie SDL, qui n'offre initialement pas ces possibilités.
Le module **Affichage 2D** regroupe quant à lui la **gestion des librairies SDL** pour permettre l'affichage à l'écran des données converties et de formes basiques bi-dimensionnelles.

2.2 Langage de programmation et bibliothèques

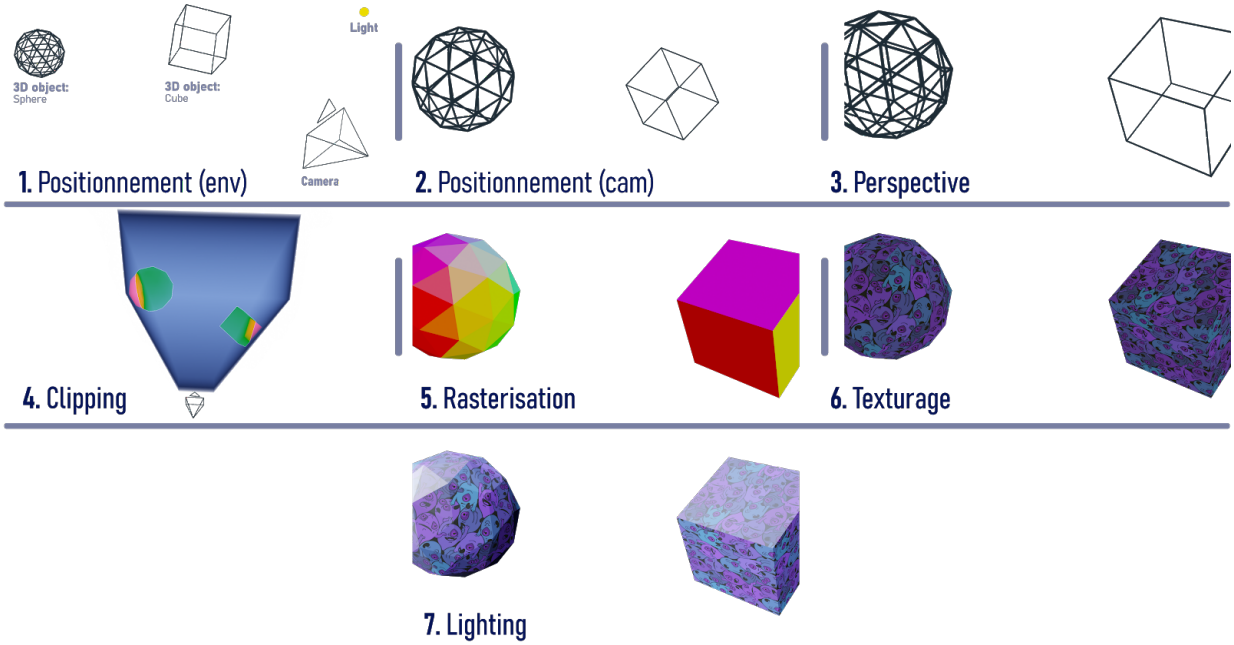
Notre projet se doit d'être optimisé pour fonctionner correctement. En effet, la modélisation tri-dimensionnelle se fait via **divers processus coûteux en ressources**, ce qui nous pousse à choisir un langage de programmation offrant un bas niveau d'abstraction. Celui-ci nous offre un meilleur contrôle sur la mémoire utilisée et les performances du programme. De plus, nous décidons d'utiliser la **librairie SDL2** [3] qui est une librairie graphique cohérente, du fait qu'elle soit optimisée, qu'elle nous laisse beaucoup de liberté et que sa popularité dans le monde de la programmation nous octroie une bonne documentation. Cette librairie graphique permettant la représentation 2D et la capture d'événements utilisateur est écrite en C.

Nous avons cependant besoin de **structurer et hiérarchiser les objets** de notre environnement 3D, mais aussi d'offrir une meilleure maintenabilité du code, dans l'objectif d'améliorer et de mettre à jour le projet. **Nous avons donc choisi le langage C++** pour son paradigme hybride : il est à la fois **procédural et orienté objet.** Le côté procédural vient du langage C, puisque le C++ intègre le C, ce qui nous permettra d'utiliser la **librairie SDL** [3]. Le côté orienté objet a rendu possible la hiérarchisation et la structuration des objets 3D, la simplification et la meilleure lisibilité pour son maintien et son évolution.

3 » Fonctionnement du moteur de rendu 3D

Notre projet propose une **interface de programmation** complète via des **méthodes génériques de création d'instances de formes 3D** pour lesquelles la gestion technique de leurs conversions pour affichage est entièrement gérée par le moteur de rendu 3D.

L'image suivante décrit par étapes le **processus du traitement d'objets 3D** via le moteur de rendu de notre projet :



À partir d'une position et d'une taille pour une forme quelconque donnée, les coordonnées de l'entité définissent sa **position dans l'environnement 3D** (1), puis par rapport à la **caméra** représentant la vue utilisateur (2), augmentés d'un **effet de perspective** contribuant au réalisme de la représentation de ces objets (3).

La phase (4) de **Clipping** [4] sert à supprimer les informations visuelles hors du champ de vision de la caméra pour améliorer les performances d'affichage. La **Rasterisation** (5) est le procédé convertissant une image vectorielle (équations mathématiques) en une image matricielle (multitude de points de couleurs, pixels) de dimension X*Y correspondant aux **dimensions de l'écran d'affichage.**

Dans les phases de **Texturing** (6) et **Lighting** (7), les faces des objets 3D sont recouverts d'une image Bitmap (2D) et rendus réalistes en faisant **varier la luminance des objets** en fonction de leur exposition aux sources de lumière.

Une fois ces **étages de pipeline** complétés, une `frame` est générée et rendue par l'interface de pro-

grammation, prête à être **incrustée dans une fenêtre d'application** basée sur notre moteur qui fournit également les classes nécessaire au **développement d'une interface graphique** élémentaire .

L'utilisation de l'ensemble des possibilités mises à disposition par notre moteur de rendu ne peuvent être apparentes uniquement **via les applications externes** que nous avons développé sur cette base.

4 » Fonctionnement des deux applications externes

4.1 Simulation d'un système stellaire

Le but de cette application est de montrer que l'on peut se servir du moteur de rendu 3D pour de la simulation scientifique. `StarSytem` permet de simuler le déplacement de 15 planètes au plus autour d'une étoile. Cette **limite de 15 planètes** a été choisie pour permettre d'observer un système stellaire moyennement grand, mais aussi pour éviter un excès de latence ainsi qu'un rendu visuel trop écoeurant dû à un surplus d'informations visuelles.

Une partie physique est implémentée pour permettre une **simulation réaliste** : nous utilisons la formule de gravitation universelle afin de calculer les forces qui s'exercent sur chaque planète. Ensuite, la **deuxième loi de Newton** permet de calculer l'accélération des planètes à partir des forces calculée précédemment. Grâce aux primitives, nous calculons la vitesse de la planète à partir de son accélération, et sa position à partir de cette vitesse. Les forces, ainsi que l'accélération, la vitesse et la position de chaque planètes sont re-calculés à chaque `tic`.

L'ensemble de ces calculs effectués sont directement visibles sur la `frame rendue de pipeline` permettant la **simulation 3D en temps réel** d'une dimension **complexe à transmettre dans un cadre éducatif.**
Une interface graphique est mise à disposition de l'utilisateur pour entrer les planètes et l'étoile qu'il souhaite puis lancer la simulation ; l'utilisation de celle-ci est détaillée dans le manuel accompagnant ce document.

4.2 Conception d'un aménagement intérieur

L'objectif de cette seconde application est de proposer une interface interactive simplifiée permettant à l'utilisateur de **visualiser, créer, modifier et charger des modèles d'aménagements intérieur.**

L'utilisateur charge ou initialise une scène à partir d'un fichier d'application ou des dimensions d'une pièce et la scène lui est alors affichée, visible depuis chacun de ses angles (vue du haut, de face, droite, gauche, arrière) avec une possibilité de déplacement libre dans la pièce via les **calculs de caméras et d'angles de vue réalisés par le moteur de rendu.**
L'utilisateur peut éditer l'ensemble caractéristiques de la scène, à savoir son nom, les papiers-peint, tapisserie et dimensions ; et peut ensuite insérer dans la pièce des **meubles préconçus** depuis deux catégories de meubles modélisés en 3D (chambre ou cuisine), ou importer n'importe quelle forme 3D depuis un fichier générique `.obj`.

Dans la pièce, l'utilisateur peut **sélectionner un meuble** et le **déplacer dans l'espace** en sachant que les **collisions entre formes** (meubles et murs) sont calculées à chaque `tic` par les classes du moteur de rendu ; en changer son orientation, son nom et son échelle.
Il est possible d'exporter une capture d'écran de la scène et de la sauvegarder en un fichier prêt à être importé pour ainsi **conférer à l'application une utilité réelle dans son domaine d'application.**

5 Difficultés rencontrées

Le domaine de recherche de notre projet est certes documenté mais reste **assez complexe** étant donné les nombreux concepts à la fois mathématiques, informatiques et relevant de l'**imagerie numérique.** En terme de programmation, certaines difficultés ont été rencontrées par rapport à la caméra utilisée pour la navigation dans un environnement tri-dimensionnel, mais aussi au niveau de la rastérisation, permettant la conversion d'une image vectorielle en une image matricielle (pouvant être affichée à l'écran).

Les **difficultés liées à la caméra** étaient dues d'une part à la recherche et prise d'informations sur les matrices de vue et de perspective et d'autre part à leur implémentation qui se devait d'être fortement optimisée. Notamment le **culling des points négatifs** d'un point de vue à la première personne a été difficile à implémenter.

Ces difficultés ont eu initialement des **conséquences contraignantes sur l'avancée des autres modules** et même bloquantes pour l'intégration de la `frame` sur les applications externes. C'est lors de la troisième semaine de programmation que le sous-module responsable de la caméra fût au point, ce déblocage nous a permis **d'ensuite enchaîner les implémentations** de l'ensemble des modules et des deux applications externes.

La rastérisation comportant des fonctions qui parcourent à minima tous les pixels de la Pipeline affichés à l'écran est une **partie critique pour les performances des applications.** Nous avons donc dû trouver un **équilibre entre maintenabilité et efficacité** ce qui n'était pas simple étant donné que les calculs se trouvent principalement dans des boucles imbriquées et que l'ordre dans lequel certaines valeurs devaient être pré-calculées était important.

Sur l'application Conception d'un aménagement intérieur, le **déplacement de meuble par clic maintenu** a montré des dépendances que nous n'avions pas prévu comme la profondeur de champ, l'orientation et l'angle de vue de la caméra ainsi que la distance du meuble à la caméra qui influent sur le vecteur de déplacement du meuble en fonction de celui de la souris.

6 » Aspect technique et projection

6.1 Des spécifications au programme final

6.1.1 Modifications et manquements

Par rapport aux spécifications initiales nous avons dû changer certains aspects du code de façon minime. Un des changements les plus flagrants est le déplacement de certaines fonctions vers le `namespace Maths`, comme par exemple les fonctions de signe et de concaténation qui se trouvaient précédemment dans le `TextureManager`. Ce choix se justifie par lui-même, étant donné le rôle du module mathématiques qui est de regrouper les tâches mathématiques lourdes.

Le **mode de déplacement** des meubles pour l'application d'aménagement intérieur a été remplacé d'un `drag and drop` par un **déplacement par touches directionnelles** pour les raisons évoquées dans la section précédente.

6.1.2 Plus-values

Le reste des modifications effectuées sont des **ajouts utiles et enrichissants.** Nous avons choisi de retirer la classe `RoundButton` car elle n'est pas utilisée tout en ajoutant les classes `RectTextButton`, `CheckBox` et

`ScrollArea` qui étaient nécessaires à un fonctionnement plus intuitif des applications externes. Bien que ces ajouts représentent des lignes de code et du travail en plus, **leur utilité reste dans les normes définies par le cahier des spécifications.**
Nous avons également ajouté une fonction supplémentaire pour pouvoir manipuler plus facilement les mouvements de la `Caméra` dans certaines situations, ainsi que des variables dans cette même classe pour faciliter l'accès aux données de la lumière dans une scène 3D.

Finalement nous avons ajouté des `bounding boxes` 2D et 3D pour les objets sur la classe `Shape` ; ceci pour simplifier l'implémentation d'une fonctionnalité avancée de l'application externe HomeDesign. Comme cette implémentation respecte les directions de notre cahier des spécifications nous n'avons pas vu d'inconvénient à l'ajouter dans la librairie de la Pipeline.

6.2 Potentielles améliorations

6.2.1 Pipeline 3D

Un des aspects majeur de la création d'une pipeline de rendu consiste en **l'optimisation de celle-ci**. En effet, la génération de `frames` étant continue, le moindre écart sur le code engendre des **conséquences drastiques sur la fluidité des applications développées** sur la librairie.

C'est pourquoi nous avons plusieurs fois analysé notre code par **profilage de l'exécution** en relevant **les étapes les plus coûteuses en ressources** pour y appliquer des optimisations et simplifications.
L'une des améliorations principales que nous aurions souhaité poursuivre serait d'**appliquer le reste des accélérations** dont nous avions déjà les ébauches mais que nous avions dû proroger pour **terminer les deux applications externes** prévues.

L'étape de traitement qui prend actuellement le plus de ressources à l'exécution est la **conversion des coordonnées 3D vers les coordonnées 2D** pour chacun des points des triangles des formes affichées à l'écran. Une optimisation logique serait de stocker dans une `map` la conversion déjà calculée pour un point 3D donné puisque **pour un maillage[2] de forme,** les triangles partagent forcément des points en commun avec leurs voisins.

6.2.2 Visualisation d'un aménagement intérieur

Cette application ayant été un pilote, **elle permet très largement la création d'une chambre** ou d'une pièce complète et les interactions avec celle-ci, mettant ainsi en avant **la manipulation d'objets 3D qu'offre notre projet** ; mais ne permet pas la conception d'une maison à plusieurs pièces non rectangulaires.

Par souci d'uniformité esthétique sur l'application, nous avons choisi de **modéliser nous même l'ensemble des 10 meubles disponibles** à l'insertion. Bien qu'il est possible d'**importer n'importe quel objet externe depuis l'application**, ce choix esthétique limite tout de même les possibilités dans une utilisation naïve. Pour un logiciel plus complet, nous aurions voulu avoir **plus de choix de meubles utilitaires et décoratifs** à la création.

6.2.3 Simulation d'un système stellaire

L'application de simulation d'un système stellaire est utilisable par l'utilisateur via une interface graphique. Cependant, cette interface pourrait être nettement améliorée avec du temps supplémentaire :

— Dans l'affichage du système, nous pourrions afficher la trajectoire de chaque planète, c'est-à-dire un tracé blanc qui ferait référence aux anciennes positions du point de gravité de chaque planète : dessiner leur trajectoire.

— Les calculs scientifiques pourraient être améliorés, notamment le fait que la variable *time* influe sur la trajectoire des planètes : en effet, cette variable désigne désigne l'intervalle de temps qui s'écoule entre chaque calcul (des nouvelles positions). Plus cette variable est petite, plus la simulation est lente, mais les trajectoire réalistes.

7 » Organisation interne et vie du groupe de projet

Une partie des membres du **groupe se passionnaient déjà pour le traitement d'images et la simulation 3D** depuis la première utilisation de la librairie SDL en première année de licence ; et ceci a constitué **la particularité de ce projet** puisque son sujet est venu de notre propre initiative et de notre propre volonté.
Ainsi, l'organisation interne et la cohésion du groupe a toujours été **animée d'un vif intérêt** pour le projet et chacune de ses applications.

7.1 Planning et répartition des tâches

Nous avons établi un **planning hebdomadaire** incluant une **répartition des tâches** par membre du groupe pour chacune des trois parties qui ont constitué ce projet. Ce partage a toujours dépendu des **préférences de chacun** tout en respectant l'**équité dans la charge de travail** et l'interdépendance des modules. Ainsi, nous nous sommes toujours divisés en sous-groupes pour **implémenter et tester chaque module isolément** ; puis réunis pour regrouper en une branche les avancées de chacun.

Notre choix d'une **gestion de projet divisée en deux chefs** s'est avérée efficace et a évité de charger davantage un seul des huit membres du groupe : nous avons fonctionné depuis le début avec **un responsable des aspects techniques,** des limites et des directions prises pour le projet, et **un délégué des aspects organisationnels,** logistiques et relationnels.

7.2 Outils et moyens techniques utilisés

— Plateforme **Discord** pour les discussions et réunions vocales ;
— **Drive partagé** entre les 8 membres du projet pour y conserver les ressources (sujet, documentations, etc.), prendre des notes partagées durant les TDs, maintenir un tableur de suivi des réunions et réfléchir ensemble pour chacune des tâches ;
— L'éditeur **Visual Studio** qui permet le code collaboratif en live et qui a grandement facilité la phase d'implémentation par sous-groupes ;
— **Github** pour le suivi des versions et la gestion en branches pour distinguer les avancements sur le cœur du moteur de rendu de ceux sur les applications externes.

7.3 Retour sur l'estimation des coûts du projet

Certains modules ont nécessité plus de temps que prévu, car il est difficile d'estimer à l'avance la difficulté d'implémentation. Inéluctablement, nous étions quelques fois confrontés à des imprévus, ce qui a modifié le temps passé sur certains modules ainsi que le nombre de ligne de code.

Module	Sous-module	Coûts estimés en amont (en heures, lignes de code)	Coûts réels du projet (en heures, lignes de code)
Géométrie et gestion des entités 3D	Gestion des entités	**8h**, 1150L	**10h,** 1750L
	Conversions géométriques	**16h**, 850L	**14h,** 600L
Module Rastérisation		**11h**, 250L	**12h,** 330L
Module Texturing		**6h**, 150L	**5h,** 200L
Outils Mathématiques		**6h**, 200L	**6h,** 500L
Affichage 2D (Gestion SDL2 pour affichages et dessins)		**4h**, 300L	**6h,** 1000L (dont 500 Draw)
Gestion de fichiers (Import .obj et export scène)		**6h**, 300L	**5h,** 300L
Interaction utilisateur (modules d'interface)		**4h**, 400L	**6h,** 950L
Application : Système stellaire		**25h**, 1000L	**28h,** 1800L
Application : Aménagement intérieur		**13h**, 700L	**20h,** 1900L
Coût total du projet		**95h**, 4900L	**112h,** 9330L

Les **écarts remarquables en lignes de code** se justifient par notre volonté de créer des classes « complètes » en traitant toujours plusieurs cas d'utilisations pour les rendre disponibles au développeur d'application externe. Ceci implique donc **plusieurs constructeurs, nombreuses surcharges de fonctions** et une prise en charge de l'affichage 2D des interface et dessins deux fois plus complète que ce que nous visions.

Vis à vis de la répartition initiale des membres sur chaque tâches, nous avons beaucoup **travaillé en paires croisées** sur les modules Géométrie, Rastérisation et Texturing afin de mieux appréhender les soucis de **compatibilité entre modules.** Nous avons également réaffecté la charge entière d'un membre **de la première vers la seconde application externe** afin d'**équilibrer le résultat** des deux avec 3 personnes par application.

8 Conclusion

Ceci clôturant notre projet de licence, l'objectif était de créer et de **mettre à disposition un moteur de rendu graphique 3D** simple d'utilisation, robuste, et **accessible au développement d'applications externes** nécessitant une visualisation ou une simulation 3D. Cet objectif **est atteint**.

Nous sommes fiers d'avoir mené à terme les deux applications **exemples d'utilisation du moteur de rendu** pour lesquelles la manipulation d'éléments 3D est primordiale et nous aurions tous espéré passer plus de temps à parachever celles-ci ; l'outil d'aménagement d'intérieur se concentrant sur **l'aspect utilitaire** d'un rendu graphique 3D (via CAO[6]) et la visualisation d'un système stellaire sur son **potentiel scientifique**.

Le module **projet** de la troisième année de licence nous a permis d'apprendre, aussi bien le travail d'équipe que de nouvelles connaissances venant de chaque membres du groupe. Notre projet est un succès, et c'est avec plaisir que nous le rendons achevé.

9 Bibliographie

Références

[1] LLC. PCMAG DIGITAL GROUP : **Graphics Pipeline**
https://www.pcmag.com/encyclopedia/term/graphics-pipeline

[2] **Maillage triangulaire** des objets 3D
https://perso.univ-rennes1.fr/pierre.nerzic/IMR2/IMR2%20-%20Synth%C3%A8se%20d'images%20-%20CM3.pdf

[3] Ubuntu : **Librairie SDL**
http://doc.ubuntu-fr.org/sdl

[4] The Ohio State University : **Clipping**
http://web.cse.ohio-state.edu/~parent.1/classes/581/Lectures/7.ClippingHandout.pdf

[5] **Culling rendering 3D**
https://docs.cryengine.com/display/SDKDOC4/Culling+Explained

[6] Futura-Sciences : Logiciel de **Conception Assistée par Ordinateur**
https://www.futura-sciences.com/tech/definitions/informatique-cao-4453/

[7] Blender Manual : **Fichier Wavefront .OBJ**
https://docs.blender.org/manual/fr/2.79/addons/io_obj.html