Un FabLab est un atelier équipé de diverses machines pilotées par ordinateur, mises à disposition du public pour la conception et la réalisation d'objets.[1] Par exemple, l'imprimante 3D permet d'« imprimer » des objets à partir de modèles 3D numériques ; la découpeuse laser permet de découper des formes complexes dans des plaques en bois, métal, plastique etc., à partir d'images vectorielles. Un des principes des FabLab est de proposer des prix plus attractifs aux utilisateurs qui laissent leur modèle (3D ou image) dans le domaine public, pour que d'autres utilisateurs puissent le réutiliser. C'est pourquoi un FabLab doit pouvoir gérer une grande collection de modèles.

Il vous est demandé de réaliser un logiciel pour la gestion d'une bibliothèque de modèles 3D pour un FabLab. Les fonctionnalités minimales requises pour le logiciel sont :

- charger un fichier qui contient un modèle 3D et l'afficher à l'écran ;
- contrôler l'affichage du modèle (tourner, rapprocher) ;
- gérer une bibliothèque de modèles 3D ;
- afficher un objet depuis la bibliothèque ;
- rechercher des objets dans la bibliothèque ;
- calculer et afficher l'image vectorielle qui correspond à une coupe d'un objet 3D.

Ce projet contribuera à vos notes de Projet tutoré, Modélisation mathématique et Conception orientée objets. Le projet est découpé en trois parties (livrables), chacune devant être rendue à une date précisée sur Moodle. Notez que pour votre note en Conception orientée objets, seul le dernier livrable sera évalué.

1 Livrable 1 : chargement et affichage d'un modèle 3D

Un modèle 3D est donné par un ensemble de segments et de triangles dans un espace vectoriel à trois dimensions. Les éléments du modèle sont bien identifiable sur l'image sur la Figure 1. Un modèle est décrit dans un fichier, suivant un format vous sera présenté en cours de Modélisation mathématique.

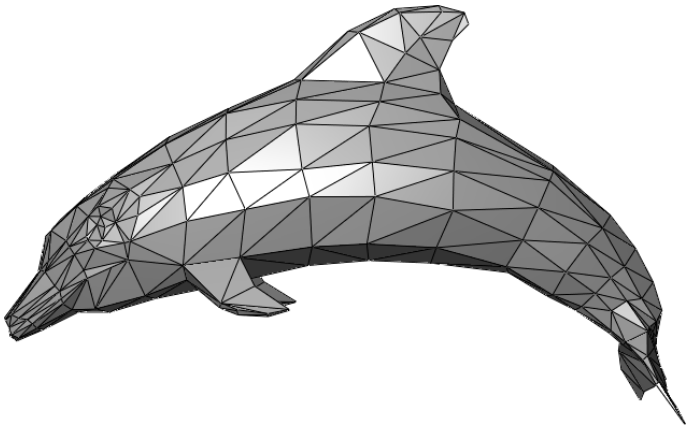

FIGURE 1 – Affichage d'un modèle 3D.

1.1 Programme exécutable

Pour le Livrable 1 vous devez faire un programme java exécutable (.jar) qui prend en paramètre un chemin de fichier, et charge un modèle 3D depuis ce fichier. Si le fichier contient des erreurs, alors le programme affichera un message d'erreur. Sinon, le programme doit ouvrir un JPanel dans lequel le modèle 3D est affiché de façon similaire à la Figure 1.

Des erreurs possibles sont : non respect du format, points ou segments manquant, etc.

Vous allez prévoir trois modes d'affichage, à choisir par une option à la ligne de commande :

- affichage des faces seulement, option `-f`
- affichage des segments seulement, option `-s`
- affichage des segments et des faces, par défaut si aucune option n'est précisée.

1. Pour plus d'information sur les FabLab,
La page FabLab de Wikipedia
Le FabLab de Lille
Exemples de réalisations au FabLab de Lille

1.2 Tests unitaires

Vous devez écrire des tests unitaires pour toutes les classes qui effectuent des calculs mathématiques. Toutes les classes de test doivent se trouver dans un dossier de code source séparé, nommé `test`, et qui est à côté du dossier `src`.

1.3 À rendre

Vous devez rendre :
- une archive jar exécutable qui permet de lancer votre programme ;
- un répertoire `data` qui contient 10 fichiers de modèles.

Le format de rendu est précisément décrit sur Moodle, et doit impérativement être respecté.

2 Livrable 2 : contrôle de l'affichage et bibliothèque de modèles

Ce livrable contiendra la plus grande partie des fonctionnalité demandées. Pour l'instant chaque fonctionnalité sera implémentée séparément, et toutes seront intégrées ensemble dans le Livrable 3.

D'une part, vous devrez permettre à l'utilisateur de contrôler l'affichage de l'objet 3D : tourner, zoomer, translater. D'autre part, vous devrez implémenter les fonctionnalités principales de la bibliothèque d'objets.

2.1 Contrôle de l'affichage

Lorsqu'un objet 3D est affiché, l'utilisateur doit pouvoir choisir l'angle de vue et le niveau de zoom.

Fonctionnalités requises L'utilisateur doit pouvoir :
- faire tourner le modèle ;
- zoomer et dézoomer ;
- translater le modèle sur l'écran ;
- le contrôle s'effectue grâce à des boutons.

Fonctionnalité optionnelle L'utilisateur peut :
- demander à ce que le modèle soit centré dans le panel d'affichage. De plus, lors du chargement et premier affichage d'un objet, celui-ci est centré.

Un modèle est dit centré si on voit la totalité de l'objet (rien ne dépasse des bords), le centre de l'objet 3D occupe le centre du panel d'affichage, et le modèle occupe bien l'espace disponible (l'objet n'est pas tout petit au centre du panel).

Programme à produire Vous ferez un programme exécutable qui prend en paramètre un chemin de fichier de modèle et ouvre une fenêtre graphique. La fenêtre contient le panel d'affichage de modèle 3D (le même que pour le Livrable 1), et un panel contenant les boutons de contrôle.

2.2 Bibliothèque de modèles

Le logiciel permet de gérer une bibliothèque de modèles 3D. Les informations de la bibliothèque sont stockées dans une base de données SQLite. Pour chaque modèle, on renseigne ces informations :
- nom du modèle, qui est un identifiant unique ;
- chemin du fichier qui contient le modèle ;
- date d'ajout à la bibliothèque ;
- ensemble de mots clés qui décrivent le modèle.

Le chemin du fichier est un chemin relatif. Tous les fichiers de modèles se trouvent dans le répertoire `data`. Ce même répertoire contient également le fichier de la base de données SQLite.

On peut également enregistrer des informations supplémentaires (optionnelles) qui vous semblent pertinentes.

Programme à produire Vous devez produire un programme exécutable qui, en fonction des options qui lui sont données, va afficher une fenêtre différente.

`--name <nom modèle>`
: fenêtre avec les informations du modèle spécifié en paramètre. Un message d'erreur approprié sera affiché si le modèle n'existe pas.

`--all`
: fenêtre avec une liste (JList) des informations sur tous les modèles de la bibliothèque.

`--find <liste mots clés>`
: fenêtre avec une liste contenant uniquement les modèles qui sont décrits par au moins un des mots clés de `<liste mots clés>`.

`--add`
: fenêtre avec formulaire de saisie des informations concernant un modèle. Un bouton permet d'ajouter le modèle à la base de données.

`--delete <nom modèle>`
: supprime de la base de données le modèle dont le nom est donné en paramètre. Le seule option qui n'ouvre pas de fenêtre.

`--edit <nom modèle>`
: fenêtre avec formulaire de saisie pour modifier les informations concernant le modèle dont le nom est donné en paramètre. Si certaines informations ne peuvent pas être changées (l'identifiant), le formulaire ne doit pas le permettre. Un bouton permet d'enregistrer les modifications dans la base de données.

L'archive que vous allez rendre (voir ci-dessous) doit embarquer une base de données non vide, stockée dans un répertoire `data`. Ce répertoire contiendra le fichier de la base de données SQLite ainsi que les fichiers de modèles. Au lancement de l'exécutable, la base de données doit être chargée et permettre de tester votre logiciel sans avoir à ajouter des modèles. Pour limiter la taille de l'archive à rendre, la base de données contiendra entre 10 et 15 modèles. Choisissez les modèles de manière à pouvoir montrer les performances de votre logiciel. Pensez à inclure tous les modèles utilisés dans la vidéo de présentation (voir ci-dessous), de manière à ce qu'on puisse reproduire cette vidéo.

2.3 Tests unitaires

Toutes les classes qui effectuent des calculs mathématiques pour l'affichage doivent être testées.

Toutes les méthodes qui manipulent la base de données doivent également être testées. Pour cela, vous allez créer une base de données dédiée aux tests, se trouvant dans un répertoire `test-data`.

Toutes les classes de test doivent se trouver dans un dossier de code source séparé, nommé `test`, et qui est à côté du dossier `src`.

2.4 Documentations diverses

Vous devez produire un **schéma UML**. Le schéma doit être *synthétique*, et aider à comprendre l'architecture du logiciel. Il doit contenir les classes principales, et seulement leurs méthodes principales, doit montrer les associations et leur donner un nom. En aucun cas le schéma UML ne doit être celui qu'on obtient par génération automatique à partir du code.

Vous devez également produire une **documentation de vos classes au format Javadoc**. Cette documentation doit être générée à l'aide de l'outil javadoc sur vos classes convenablement commentées.

Finalement, vous devez faire une **vidéo de présentation des fonctionnalités 3D**. La vidéo durera 2-3 minutes et sera faite par capture d'écran. Elle doit illustrer l'utilisation du programme d'affichage spécifié dans la Section 2.1. La video peut être sous-titrée ou contenir des explications parlées. Vous veillerez à présenter votre logiciel sous son meilleur jour, et illustrer ses performances par exemple en affichant des modèles volumineux et/ou des modèles de forme irrégulière difficiles à centrer.

2.5 À rendre

Vous devez rendre :

- une archive jar exécutable qui lance le programme d'affichage 3D spécifié dans la Section 2.1, et qui contient tous les fichiers source ;
- une archive jar exécutable qui lance le programme de manipulation de la base de données spécifié dans la Section 2.2
- un répertoire [`data`] qui contient une base de données SQLite non vide et des fichiers de modèles de la base ;
- une vidéo de présentation ;
- un rapport d'une à deux pages qui décrit comment chaque membre de l'équipe a contribué à la réalisation du livrable ;
- un schéma UML ;
- la documentation au format Javadoc.

Le format de rendu est précisément décrit sur Moodle, et doit impérativement être respecté.

3 Livrable 3 : fenêtre d'application et fonctionnalités avancées

Pour ce livrable, vous allez intégrer toutes les fonctionnalités dans une seule fenêtre d'application. Vous allez également ajouter des fonctionnalités plus complexes liées à l'affichage et aux problématiques d'un FabLab.

3.1 Refactoring MVC

Vous allez commencer par réorganiser votre code pour respecter le patron de conception Modèle–Vue–Contrôleur (MVC). Identifiez les packages qui contiendront les classes du modèle. Créez un package pour les vues, et éventuellement un package pour les contrôleurs. Organisez vos classes dans les différents packages, cela va probablement nécessiter de décomposer certaines classes pour séparer les différentes préoccupations (modèle, vue, contrôleur).

3.2 Fenêtre d'application

Toutes les fonctionnalités déjà implémentée doivent être intégrées dans une unique fenêtre d'application. Vous devez concevoir la fenêtre de l'application qui contiendra les différents panels implémentés pour le Livrable 2.

Optionnellement, vous pouvez ajouter la possibilité de contrôler l'affichage grâce à la souris et au clavier (zoom, translation, tourner, etc.)

3.3 Vue en coupes

Il s'agit d'une fonctionnalité utile dans un FabLab. Une technique souvent utilisée est de construire un objet en collant des tranches, comme sur la Figure 2.

FIGURE 2 – Objet construit en collant des tranches de bois.

Pour le faire, on doit pouvoir découper des tranches de la bonne forme, et donc pouvoir connaître la forme des tranches. Vous devez implémenter le calcul de la forme des tranches.

Concrètement, l'utilisateur doit pouvoir choisir un modèle dans la bibliothèque et demander son découpage en un nombre de tranches qu'il va spécifier. Le découpage se fera suivant le plan parallèle au plan d'affichage, d'avant en arrière. Le logiciel devra alors afficher la première tranche, et permettre de naviguer entre les tranches en avant et en arrière.

Note : une tranche est définie par un ensemble de segments, qui sont les intersections des triangles du modèle 3D avec le plan de coupe.

3.4 Éclairage, lissage

Pour l'instant l'affichage de modèles 3D se fait en coloriant chaque face (triangle) avec une couleur uniforme, ce qui donne un affichage 3D rudimentaire. Il vous ai demandé d'implémenter deux techniques qui permettent d'améliorer la qualité d'affichage, et qui vous seront expliquées en cours de Modélisation mathématique.

Éclairage (fonctionnalité obligatoire) : on doit définir une source de lumière, qui est un point dans l'espace ; vous prendrez un point fixé, par exemple à 45 degrés vers la droite de l'observateur. La couleur de chaque face dépend alors de l'angle entre les rayons qui proviennent de la source de lumière et le plan de la face.

Lissage (fonctionnalité optionnelle) : comme précédemment, il y a une source de lumière, mais la couleur d'une face n'est pas uniforme. Sur chaque face on affiche un gradient de couleur, qui dépend de l'éclairage des angles du triangle. Cette méthode permet d'avoir un affichage 3D de très bonne qualité avec relativement peu de triangles, mais elle est plus gourmande en calcul. À tester avec des objets avec peu de triangles d'abord.

3.5 Documentations diverses

Comme pour le Livrable 2, vous devez produire un schéma UML, en respectant les mêmes contraintes, ainsi qu'une documentation Javadoc.

Vous devez aussi refaire une vidéo de présentation de **4 à 6 minutes**, qui maintenant présentra toutes les fonctionnalités du logiciel (affichage, manipulations de la base de données, fonctionnalités supplémentaires).

Vous rapporter à la Section 2.4 pour plus de détails.

3.6 Barème indicatif pour la note pratique de Conception orientée objets

Votre note pratique de Conception orientée objets sera donnée par l'évaluation de ce dernier livrable uniquement. La note prendra en compte les aspects suivants de votre travail (le barème est indicatif et susceptible d'être modifé) :

- le respect et l'implémentation correcte du MVC (6 points)
- la qualité de l'UML (4 points)
- la mise en oeuvre des tests (boite blanche et/ou boite noire) (4 points).
- la documentation Javadoc (3 points)
- l'appréciation globale de l'efficacité de votre code (3 points)

3.7 À rendre

Vous devez rendre :

- une archive jar exécutable qui lance le programme ;
- un répertoire [`data`] qui contient une base de données SQLite non vide et des fichiers de modèles de la base pour tester ;
- une vidéo de présentation ;
- un rapport d'une à deux pages qui décrit comment chaque membre de l'équipe a contribué à la réalisation du livrable ;
- un schéma UML ;
- la documentation au format Javadoc.

Le format de rendu est précisément décrit sur Moodle, et doit impérativement être respecté.

4 Objectifs, exigences, organisation du travail

Ce projet tutoré poursuit plusieurs objectifs pédagogiques.

D'abord, il y a des objectifs techniques : pratique de la programmation java, utilisation de modèles mathématiques, utilisation de nouvelles bibliothèques (SQLite), utilisation d'un gestionnaire de versions (git).

Nous visons également des apprentissages non techniques, mais tout aussi importants. Le projet est relativement conséquent et sera fait en groupes de 4 à 6 personnes dont la composition est imposée. Le mener à bien demandera de bien s'organiser, se coordonner, communiquer. Ainsi, la note de projet tiendra compte non seulement de la qualité technique du logiciel réalisé, mais aussi de votre capacité à vous organiser pour mener ce travail.

Finalement, testons votre capacité à utiliser un spécification écrite (ce document). Ceci implique la compréhension complète et non ambigue du document : relisez le document souvent, mettez en doute votre compréhension, posez des questions !

D'après le programme du DUT, ce projet demande 50 heures de travail individuel par étudiant, soit 3 heures par semaine en moyenne. C'est à chacun d'entre vous de s'assurer qu'il fait sa partie du travail.

L'utilisation du git de l'IUT est obligatoire . Pour éviter la triche, votre dépôt git doit être privé, visible uniquement par les membres du projet et vos enseignants. En cas de triche vous serez tenus responsables, même si quelqu'un a récupéré votre travail à votre insu.

Travail régulier et équitable, notation La note de projet tiendra compte de :

- la régularité du travail : un travail régulier témoigne d'une bonne organisation et une bonne coordination et sera récompensé. Vous aurez une meilleure note si vous travaillez sur le projet toutes les semaines (par rapport à tout faire la semaine qui précède la date de rendu) ;
- l'implication équitable de tous les membres du groupe : réussir à impliquer chacun quel que soit son niveau est une compétence très importante, et sera récompensée.

Voici quelques règles à respecter pour permettre à l'enseignant de bien évaluer les points ci-dessus.

- commitez votre code vous même et au plus tôt ;
- mettez sous git tous les documents produits (rapport, maquettes, schémas), pas seulement le code ;
- si une contribution n'est pas visible dans git, la faire figurer dans le rapport.

Toute contribution sera prise en compte. Cependant, un étudiant ne peut pas se consacrer uniquement à des tâches qui ne nécessitent pas de programmer. Chacun doit contribuer en écrivant du code *original*.[2] Il n'est pas demandé que tous les membres d'un groupe écrivent la même quantité de code, mais il ne doit pas y avoir de grands déséquilibres.

Si vous avez contribué avec quelque chose qui n'est pas visible dans git (par ex. recherche de modèles 3D, capture de la vidéo de présentation, préparation de l'archive à rendre, etc.), assurez vous que c'est mentionné dans le rapport.

En cas de **conflits dans le groupe, en parler** à l'enseignant de TP au plus vite ; il/elle pourra jouer le rôle de médiateur pour vous aider à résoudre le problème.

Fonctionnalités requises et optionnelles Un marteau en bois ne permet pas d'enfoncer un clou, même s'il a été sculpté et gravé par le grand maître Paul Gauguin. Pour enfoncer un clou, il suffit d'un marteau ordinaire pourvu que sa tête soit en fer.

Ne faites pas un logiciel qui ressemble à un marteau à tête en bois.

Autrement dit, il faut concentrer vos efforts et donner une grande priorité aux fonctionnalités exigées par le sujet. Seulement après vous pourrez vous faire plaisir en implémentant des fonctionnalités supplémentaires, ou embellir votre logiciel.

2. Effacer du code, faire un commit, puis remettre le code effacé et refaire un commit pour faire semblant d'avoir contribué, ça se voit dans git !