M2107 - Projet de Java - Maillages triangulaires

Rémi Synave

Contexte

Ce projet vous permettra de consolider vos acquis en Java et d'assimiler le concept de tableau dynamique.

Le thème du projet est un domaine tout à fait passionnant : **les maillages triangulaires**. Vous pouvez relire ces derniers mots en vous imaginant une musique épique pour plus d'effet.

A priori, peu de chance que vous connaissiez ce domaine. Pas de panique ! Tout ce dont vous aurez besoin sera expliqué dans ce document. De plus, un morceau de "cours" un peu plus détaillé est disponible sur Moodle. Il est recommandé de consulter ce document avant de commencer réellement le projet.

Le travail sera à réaliser seul ou en binôme (pas de trinôme). Les membres d'un binôme peuvent être dans des groupes différents. A l'emploi du temps, vous devez avoir 21h de TP pour réaliser ce projet. Vous êtes en autonomie complète. Toutefois :

— si vous êtes bloqués, vous pouvez demander de l'aide par mail à remi.synave@univ-littoral.fr (joignez votre code) ou sur discord.

— l'appel aurait pu être fait à n'importe quel moment. Vous auriez dû respecter les salles mais ce n'est plus vraiment d'actualité...

Le projet est décomposé en 2 parties. Seule la première est obligatoire. Pour les plus rapides ou les plus curieux, vous pouvez faire la partie 2.

Dans tous les cas, en plus du code produit, vous devrez rendre un mini rapport. Il devra comporter les quelques réponses aux questions posées. Vous y noterez également toutes les remarques importantes du développement. **Finalement, il devra faire état de l'avancement final, c'est à dire, de ce qui a été fait ou non. Pour tout ce qui n'a pas été fait, vous ferez une analyse de la cause du non développement.**

L'évaluation portera sur le code que vous allez produire ainsi que le mini rapport qu'il vous est demandé d'écrire.

TODO : Dans votre répertoire de travail, décompressez l'archive que vous avez récupéré sur Moodle. Vous y trouverez un répertoire `M2107ProjetJava`. Toutes les classes et votre mini rapport devront se trouver dans ce répertoire de travail qui sera à compresser et rendre à sur Moodle.

Modèle numérique - Maillage triangulaire

Un modèle numérique est la représentation numérique d'un objet. Il existe 4 familles de modèles numériques : les modèles volumiques continus, les modèles volumiques discrets, les modèles surfaciques continus et les modèles surfaciques discrets.
Ces représentations sont utiles dans des domaines aussi variés que : les jeux vidéos, le cinéma, l'architecture, la médecine, etc.

Le maillage triangulaire est une représentation surfacique discrète d'un objet :
— Surfacique car seule la surface de l'objet est modélisée.
— Discrète car elle est approximée par un ensemble fini d'éléments.

Comme son nom l'indique, le maillage triangulaire représente la surface d'un objet grâce à des triangles. La surface peut être couverte d'une texture pour un rendu réaliste de l'objet. Dans ce projet, nous ne nous intéresserons qu'à la géométrie de l'objet. Bien qu'il soit possible de créer beaucoup de petits triangles pour représenter la surface le plus fidèlement possible, celle-ci reste approximée (voir figure 1). Seules les surfaces planes (assez rare dans les objets réels) sont modélisables de manière exacte.

Les triangles sont formés par 3 sommets eux mêmes caractérisés par une position `(x,y,z)` dans l'espace. Il existe de nombreuses méthodes pour stocker les informations d'un maillage triangulaire. Nous nous intéresserons ici à la structure de données suivante :

— Des sommets stockés dans un tableau. Chaque sommet est caractérisé par ses 3 coordonnées (x,y,z). Il est aussi identifié par un indice (le numéro de la case du tableau). De plus, chaque sommet aura une liste de sommets voisins. Les voisins seront identifiés par leur indice dans le tableau.
— Des faces stockées dans un tableau. Chaque face est caractérisée par les 3 indices des sommets formant le triangle. Chaque face est également identifée par un indice (le numéro de la case du tableau).

Un exemple de maillage triangulaire ainsi que ses données sous forme de tableaux sont donnés en figure 2 et 3.

Remarque : les arêtes ne sont pas stockées explicitement mais peuvent être retrouvées en parcourant les faces. Chaque face étant caractérisée par 3 indices de sommet, il est facile de recréer l'ensemble des arêtes. Chaque arête est définie par les indices de 2 sommets. Par exemple, pour la face 3 de la figure 2, celle-ci est définie par les sommets 5, 3 et 7. Les arêtes entourant cette face sont les arêtes `(5,3)` ou `(3,5)`, `(7,3)` ou `(3,7)` et `(7,5)` ou `(5,7)`.

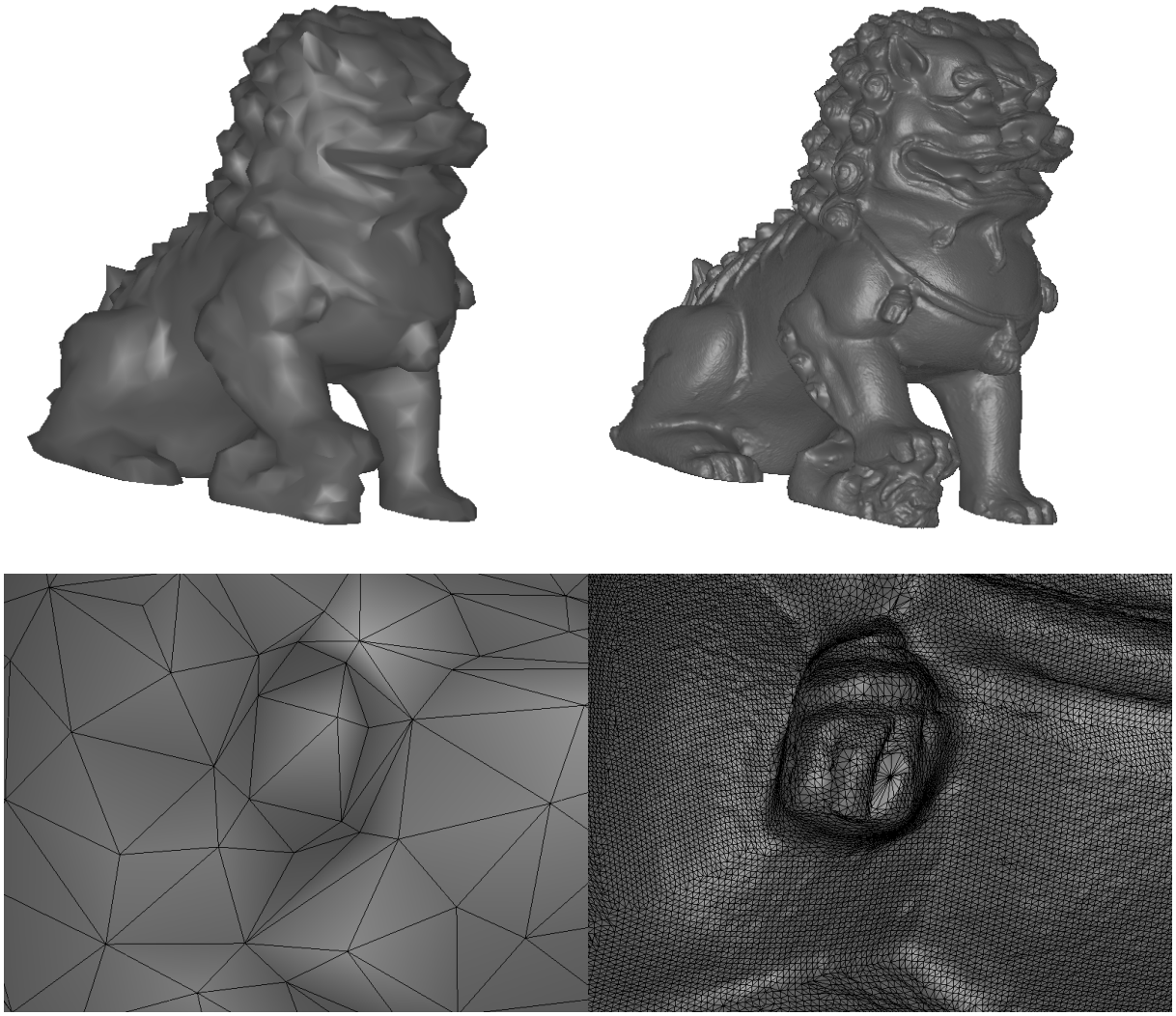

FIGURE 1 – Modèle `783-Chinese_dragon` issu du dépôt AIM@SHAPE-VISIONAIR. En haut : maillage triangulaire à deux niveaux de détails différents. Grossier à gauche et détaillé à droite. En bas : zoom sur la clochette au dessus de la patte droite du dragon.

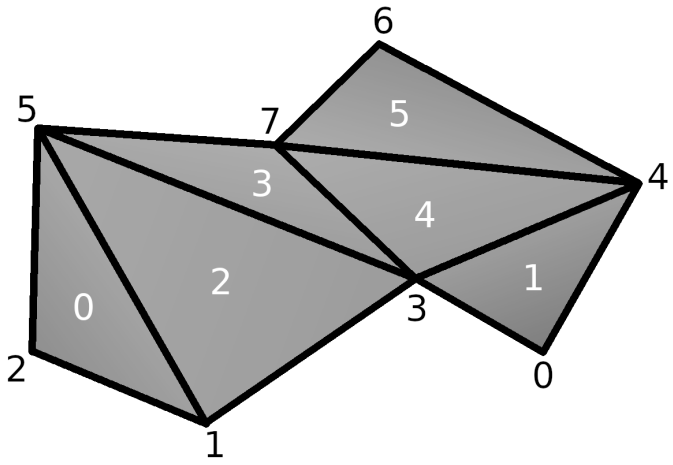

FIGURE 2 – Exemple de maillage triangulaire. Les chiffres noirs donnent les numéros d'indice des sommets. Les chiffres blancs donnent les numéros d'indice des faces. Les tableaux associés à ce maillage sont donnés en figure 3.

Tableau de sommets

Numéro du sommet	Coordonées X	Coordonées Y	Coordonnées Z	Liste des voisins
0	4.5	1.2	-0.5	(3,4)
1	1.8	0.5	-0.7	(2,5,3)
2	0.5	1.1	0.03	(5,1)
3	3.2	1.9	0.46	(1,5,7,4,0)
4	5	3	0.5	(6,7,0,3)
5	0.7	3.2	-0.4	(2,1,3,7)
6	3	3.9	-0.28	(7,4)
7	2.2	3.1	-0.61	(5,3,4,6)

Tableau de faces

Numéro de la face	Indice du sommet 1	Indice du sommet 2	Indice du sommet 3
0	2	1	5
1	0	4	3
2	1	3	5
3	5	3	7
4	3	4	7
5	7	4	6

FIGURE 3 – Structure de données associée au maillage triangulaire de la figure 2.

MG3D

MG3D est une bibliothèque encore en développement. Elle permettra de faciliter le développement d'applications graphiques 3D. Vous allez utiliser (une sous-partie de) cette bibliothèque afin d'atteindre l'objectif obligatoire (partie 1) et celui de la quête annexe (partie 2).

MG3D permet de manipuler les maillages triangulaires. Il existe une grande variété de formats de fichier pour stocker la géométrie des maillages triangulaires (stl, obj, fbx, wrl, ply, etc.) ainsi que des données supplémentaires comme les textures ou encore les lumières. Le format de fichier reconnu par MG3D est le `off`. Pour ce projet, des maillages de travail vous sont fournis mais d'autres peuvent être téléchargés sur le dépôt suivant :
`http://visionair.ge.imati.cnr.it/ontologies/shapes/`

La structure de données utilisée par MG3D est celle décrite dans la section précédente.

Les points les plus importants de la bibliothèque que vous devrez utiliser dans ce projet sont décrits ci-après. Pour plus de détails, vous pouvez vous reporter à la documentation fournie dans l'archive de MG3D.

La classe `Maillage` permet d'ouvrir un fichier `off` et d'accéder à toutes ses propriétés : nombre de sommets, nombre de faces, position d'un sommet, voisins d'un sommet et sommets d'une face.

A partir d'ici, le code donné en exemple fonctionne si et seulement si la bibliothèque MG3D a été correctement installée. Ce sera d'ailleurs la première étape du projet.

Les maillages triangulaires de travail

Les maillages triangulaires mis à votre disposition sont fournis sous deux formats : `off` et `ply`. Le format `off` est reconnu par MG3D. C'est donc bien le fichier avec cette extension qui doit être utilisé avec MG3D.

Le même fichier est fourni avec l'extension `ply`. Il représente le même maillage triangulaire. Ce format de fichier est reconnu par `Blender`. Ces fichiers peuvent donc y être importé directement. Toutefois, si le coeur vous en dit, vous pouvez ajouter un add-on à `Blender` afin de pouvoir importer les fichiers `off` :
`https://github.com/alextsui05/blender-off-addon`

`Blender` vous permettra de visualiser les maillages triangulaires que vous allez manipuler.

Utiliser la bibliothèque MG3D

Pour utiliser (les classes de) la bibliothèque MG3D dans votre code, vous devrez importer les classes utiles. Vous aurez donc sûrement besoin de :

```
import MG3D.geometrie.Maillage;
```

ou encore

```
import MG3D.geometrie.Sommet;
```

ou, si vous pensez avoir besoin de toutes les classes :

```
import MG3D.geometrie.*;
```

Rappel : L'import est à ajouter dans votre fichier avant même la déclaration de votre classe.

Charger un maillage contenu dans un fichier `off`

```
Maillage m = new Maillage("monFichier.off");
```

Dans la suite du document, le maillage sera toujours noté `m`.

Accéder aux nombres de sommets et de faces d'un maillage

```
int nbSommets=m.getNbSommets();
int nbFaces=m.getNbFaces();
```

Accéder au i^{eme} sommet du maillage

`i` est l'indice du sommet auquel on veut accéder. `i` a été déclaré et initalisé.

```
Sommet s=m.getSommet(i);
```

Récupérer les coordonnées d'un sommet

`s` est un sommet.

```
double x=s.getX();
double y=s.getY();
double z=s.getZ();
```

Connaitre le nombre de voisins du i^{eme} sommet du maillage

`s` est le sommet pour lequel on veut connaitre le nombre de voisins.

```
int nbVoisins=s.getNbVoisins();
```

Connaitre le numéro d'indice du i^{eme} voisin d'un sommet

`s` est le sommet pour lequel on veut connaitre le numéro d'indice du i^{eme} voisin.

```
int indiceSommet=s.getVoisin(i);
```

Accéder à la i^{eme} face du maillage

`i` est l'indice de la face à laquelle on veut accéder. `i` a été déclaré et initalisé.

```
Face f=m.getFace(i);
```

Récupérer les numéros d'indice des trois sommets composant une face

`f` est la face pour laquelle on veut connaitre les indices des sommets la formant.

```
int indiceS1=f.getS1();
int indiceS2=f.getS2();
int indiceS3=f.getS3();
```

Récupérer la longueur entre deux sommets

`s1` et `s2` sont les numéros des deux sommets pour lesquels on veut avoir la distance les séparant.

```
double longueur=m.distanceEuclidienneEntreSommets(s1,s2);
```

Partie 1 - Vérifier l'imprimabilité d'un modèle numérique

TODO : Avant tout et si cela n'a pas encore été fait, téléchargez et installez la bibliothèque MG3D. Cette bibliothèque ne doit pas être modifiée durant le projet ! Les classes que vous allez écrire utiliseront MG3D mais vous ne devez pas modifier le code source de la bibliothèque.

Le but de ce projet est de vérifier si un modèle numérique (un maillage triangulaire) peut être reproduit grâce à une imprimante 3D. Un maillage triangulaire imprimable doit respecter un certain nombre de contraintes (voir diapo de cours pour plus de détails) :

— Avoir une surface fermée, c'est à dire une surface continue sans bord.
— Ne pas contenir d'auto-intersection.
— Ne pas contenir de triangle dégénéré.

Afin de simplifier le problème, nous ferons l'hypothèse que les triangles présents dans les modèles numériques à votre disposition ne sont pas dégénérés. De plus, dans un fichier, un seul objet sera représenté. Un fichier ne contient pas plusieurs objets disjoints.

Dans un premier temps, vous allez mettre en place les outils nécessaires à la vérification des conditions de fermeture de la surface et de l'absence d'auto-intersection. Ensuite, vous utiliserez ces outils pour écrire un programme permettant de savoir si un maillage triangulaire est imprimable ou non.

Création de la classe `Arete`

Comme vous avez pu le constater lors de la description de la structure de données, les arêtes ne sont pas sotckées dans les tableaux décrivant un maillage triangulaire. Pourtant, afin de vérifier que la surface est bien fermée, vous aurez besoin du nombre d'arêtes. Attention ! Le nombre d'arêtes n'est pas égal à 3 fois le nombre de faces (une face étant entourée par 3 arêtes). En effet, certaines arêtes sont partagées par plusieurs faces.

TODO : Dans votre répertoire `M2107ProjetJava`, développez la classe `Arete`. Une arête est définie par deux (indices de) sommets. Pour cette classe, prenez exemple sur la classe `Face` de MG3D. Une attention particulière sera portée à la méthode `equals`. **Attention**, une arête définie par les sommets 1 et 2 et celle définie par les sommets 2 et 1 correspondent à une seule et même arête. Si de telles arêtes sont comparées, la méthode `equals` doit retourner `true`.

TODO : Dans la classe `Face`, vous verrez que la méthode `equals` n'est pas tout à fait comme décrit dans le cours. Dans le mini rapport, expliquez pourquoi ce qui a été dit durant le cours était approximatif. N'hésitez pas à donner des liens vers des pages web pour appuyer vos dires.

Génération des arêtes d'un maillage triangulaire

TODO : Dans la classe `MainPartie1`, développez une méthode permettant de génèrer la liste des arêtes. Cette méthode prendra en paramètre un objet de type `Maillage` et devra retourner un tableau dynamique contenant les arêtes sans doublon (objet de type

`ArraysList<Arete>`). Évidemment, le tableau retourné devra être rempli avec les arêtes du maillage passé en paramètre.

L'algorithme pour la génération est très simple :

```
Algo 1 :
m désigne le maillage triangulaire
-------------------------------------------------------------------------
Créer le tableau dynamique d'arêtes
Pour toutes les faces f de m
  Récupérer les sommets s1, s2 et s3 de f
  Si l'arête (s1,s2) n'est pas dans le tableau dynamique alors
    Ajouter l'arête (s1,s2) dans le tableau dynamique
  Fin Si
  Si l'arête (s1,s3) n'est pas dans le tableau dynamique alors
    Ajouter l'arête (s1,s3) dans le tableau dynamique
  Fin Si
  Si l'arête (s2,s3) n'est pas dans le tableau dynamique alors
    Ajouter l'arête (s2,s3) dans le tableau dynamique
  Fin Si
Fin pour
retourner le tableau dynamique
```

Vérification de la fermeture de la surface du maillage triangulaire

TODO : Dans votre classe `MainPartie1`, écrivez une méthode permettant de savoir si la surface de l'objet est bien fermée. La méthode prendra en paramètre un objet de type `Maillage` et retournera `true` si la surface est fermée, `false` sinon.
Indice pour écrire la méthode : `V-E+F=2`

TODO : Dans votre mini rapport, décrivez la méthode que vous avez mise en place pour vérifier cette caractéristique. N'hésitez à mettre des liens vers des pages web sur lesquelles vous vous êtes appuyés.

Auto-intersection

Une auto-intersection est présente dans un maillage triangulaire lorsque deux triangles définissant la même surface s'intersectent. Lorsque deux triangles s'intersectent, une arête d'un triangle passe "à l'intérieur" de l'autre triangle.

Algorithme de `Möller-Trumbore`

L'algorithme d'intersection de `Möller-Trumbore` permet de savoir si un rayon (une arête dans notre cas) intersecte un triangle (une face dans notre cas).

TODO : Dans la classe `MainPartie1`, développez la méthode `MollerTrumbore` permettant de vérifier si un rayon intersecte un triangle. La signature de la méthode est déjà présente dans le fichier et le pseudo-code, adapté à notre situation, vous est donné pour vous aider. Aucune bibliothèque annexe n'est nécessaire. Il est inutile de comprendre les principes mathématiques pour programmer cet algorithme (mais c'est mieux si vous pouvez). Avant de vous lancer, étudiez le documentation des classes `Point3D`, `Sommet` et `Vecteur3D` de `MG3D`.

```
Algo 2 :
origineRayon désigne le point d'où part le rayon
rayon désigne le vecteur donnant la direction du rayon
sommet0 désigne le premier sommet du triangle
sommet1 désigne le second sommet du triangle
sommet2 désigne le troisième sommet du triangle
------------------------------------------------------------------------
EPSILON=0.0000001
arete1 = vecteur partant d'origine sommet0 et d'extrémité sommet1
arete2 = vecteur partant d'origine sommet0 et d'extrémité sommet2
h = produit vectoriel de rayon et arete2
a = produit scalaire de arete1 et h
Si a>-EPSILON et a<EPSILON alors
  retourner faux
Fin Si
f = 1/a
s = vecteur partant de sommet0 et d'extrémité origineRayon
u = f*(produit scalaire de s et h)
Si u<0 ou u>1 alors
  retourner faux
Fin Si
q = produit vectoriel de s et arete1
v = f*(produit scalaire de rayon et q)
Si v<0 ou u+v>1 alors
  retourner faux
Fin Si
t = f*(produit scalaire de arete2 et q)
Si t>EPSILON et t<(1-EPSILON) alors
  retourner vrai
Fin Si
retourner faux
```

Algorithme de vérification de la présence d'auto-intersection

TODO : Dans votre classe `MainPartie1`, écrivez la méthode permettant de savoir si un maillage triangulaire possède au moins une auto-intersection. La méthode prendra en paramètre un objet de type `Maillage` et retournera `true` si au moins une auto-intersection a été détectée, `false` sinon. L'algorithme est donné ci-dessous. Toutefois, faites attention de bien comprendre le rôle des paramètres de l'algorithme de `Möller Trumbore`.

Pour vérifier la présence d'une auto-intersection dans un maillage triangulaire, l'algorithme est le suivant :

```
Algo 3 :
m désigne le maillage triangulaire
-------------------------------------------------------------------------
Générer la liste des arêtes
Pour toutes les arêtes a de m
  Pour toutes les faces f de m
    Si MollerTrumbore entre l'arête a et la face f retourne vrai alors
      retourner vrai
    Fin Si
  Fin Pour
Fin Pour
retourner faux
```

Programme principal

TODO : Toujours dans votre classe `MainPartie1`, écrivez un programme principal permettant de savoir si un maillage triangulaire est imprimable ou non. Le nom du fichier contenant le maillage triangulaire sera passé en argument au programme lors de son lancement. Le programme affichera, en fonction du maillage, soit :

- `Le maillage triangulaire est imprimable.`
  ou
- `Le maillage triangulaire n'est pas imprimable car la surface n'est pas fermée.`
  ou
- `Le maillage triangulaire n'est pas imprimable car il présente au moins une auto-intersection.`
  ou
- `Le maillage triangulaire n'est pas imprimable car la surface n'est pas fermée et il présente au moins une auto-intersection.`

TODO : Dans votre mini rapport, donnez le résultat de l'imprimabilité pour tous les maillages triangulaires qui vous ont été fournis. Certains maillages peuvent nécessiter un long temps de calcul (plusieurs minutes pour `bunny`).

A l'aide de `Blender`, visualisez les modèles `elephant-50kv`, `heptoroid` et `rocker-arm`. Pensez vous vraiment que la surface n'est pas fermée ? Que se passe t-il ? Répondez dans votre mini rapport. N'hésitez pas à donner des liens vers des pages web qui illustrent vos dires.

Question subsidiaire

Trouvez des noms de structures de données permettant de stocker des maillages triangulaires et décrivez les rapidement dans votre mini rapport. N'hésitez pas à donner des liens vers des pages web qui en parlent.

Partie 2 - Calculer des distances sur le modèle numérique

Attention ! Pour cette partie, n'utilisez pas les maillages `bunny`, `elephant-50kv` ni `heptoroid`. Les calculs seraient beaucoup trop longs ! MG3D n'a pas été optimisé pour de tels volumes de données.

L'objectif de cette partie est de pouvoir prendre des mesures sur un modèle numérique. Vous allez programmer une méthode de calcul de distance entre deux points en suivant la surface de l'objet.

Avant tout, si vous voulez pouvoir définir vous-mêmes les sommets de départ et d'arrivée de la mesure, vous aurez besoin de connaitre les numéros d'indice des sommets des maillages triangulaires. Pour accéder à cette information, vous pouvez consulter la page web suivante : `http://www-lisic.univ-littoral.fr/~synave/autres/blender/blender.html`

Nous allons partir de deux principes :
— Un maillage triangulaire est un graphe.
— Le chemin le plus court entre deux points est la ligne droite.

Dijkstra est le nom d'un informaticien Néerlandais qui a publié un algorithme permettant de trouver le plus court chemin entre deux noeuds (ou sommets) dans un graphe. Cet algorithme correspond exactement à ce que nous cherchons !

Le pseudo-code de l'algorithme de Dijkstra est donné ci-dessous :

```
Algo 4 :
m désigne le maillage triangulaire
sommet_source désigne le numéro du sommet de départ
sommet_destination désigne le numéro du sommet d'arrivée
-------------------------------------------------------------------------
Pour tous les sommets s de m
  s.parcouru=infini
  s.precedent=0
Fin Pour

pas_encore_vu=liste de tous les sommets
sommet_source.parcouru=0

Tant que pas_encore_vu est non vide
  on cherche le sommet s1 ayant la distance parcourue la plus petite
  on enleve ce sommet s1 de la liste
  Pour tous les sommets s2 voisins de s1
    Si s2.parcouru>s1.parcouru+distance(s1,s2)
      s2.parcouru=s1.parcouru+distance(s1,s2)
      s2.precedent=s1
    Fin Si
  Fin pour
Fin Tant que

chemin=liste vide
s=sommet_destination

Tant que s n'est pas le sommet_source
  ajouter le sommet s au début de la liste chemin
  s=s.precedent
Fin Tant que

ajouter le sommet_source au début de la liste chemin

retourner le chemin
```

Cet algorithme retourne un tableau dynamique d'`Integer` dans lequel sont stockés tous les numéros d'indice des sommets par lesquels il faut passer pour aller du point de départ de prise de la mesure au point d'arrivée.

Connaissant le chemin complet, il est simple de calculer toutes les distances séparant chaque étape du chemin et en faire l'addition pour avoir la distance totale entre les deux

points en suivant la surface de l'objet.

TODO : Dans la classe `MainPartie2.java`, vous pouvez étudier la méthode `Dijkstra`. Cette méthode est commentée mais possède des trous dans son code. Dans un premier temps, en vous aidant du pseudo-code, vous allez compléter les parties manquantes.

TODO : Dans la classe `MainPartie2`, créez un programme principal permettant d'afficher la distance entre deux points sur un maillage. Afin de vérifier le bon fonctionnement de votre programme, comparez vos résultats aux suivants :

Nom du modèle	Sommet de départ	Sommet d'arrivée	longueur
cube.off	0	7	2.414
moomoo.off	15	825	38.868
moomoo.off	3355	164	42.677
moomoo.off	3333	574	33.635
sharp_sphere.off	25	5000	27.064
sharp_sphere.off	361	852	28.011
rocker-arm.off	985	1649	47.380
rocker-arm.off	6273	5727	26.258
mannequin.off	115	203	2.904
mannequin.off	239	332	2.483

Si votre programme fonctionne, il est temps de vous confronter à la réalité : scannez un objet et vérifiez que votre algo donne sensiblement la même mesure qu'une mesure manuelle sur l'objet réel ou au moins une valeur de même ordre de grandeur.

Pour ce faire, il vous faudra demander l'accès au scanner 3D au meilleur enseignant de tous les IUT confondus. (Non, il ne s'agit pas de M. Capitaine ! Bande de rigolos ! ! !)
→Évidemment infaisable si nous ne sommes pas à l'IUT.

TODO : Dans votre mini rapport, faites un comparatif entre les mesures manuelles sur l'objet réel et les mesures sur le modèle numérique.

Normalement, les mesures prises sur l'objet réel doivent être plus petites que celles trouvées sur le modèle numérique. En effet, lors de l'acquisition au scanner 3D, les triangles modélisant la surface ne permettent pas de trouver un chemin tout droit entre les deux points choisis. Sur le modèle numérique, le chemin entre les deux points zigzaguent sur la surface. (voir la figure 4).

TODO : Dans votre mini rapport, imaginez et proposez une solution pour régler ou amoindrir ce problème.

Si vous avez terminé

Il y a encore plein de choses à faire ! N'hésitez pas à demander.

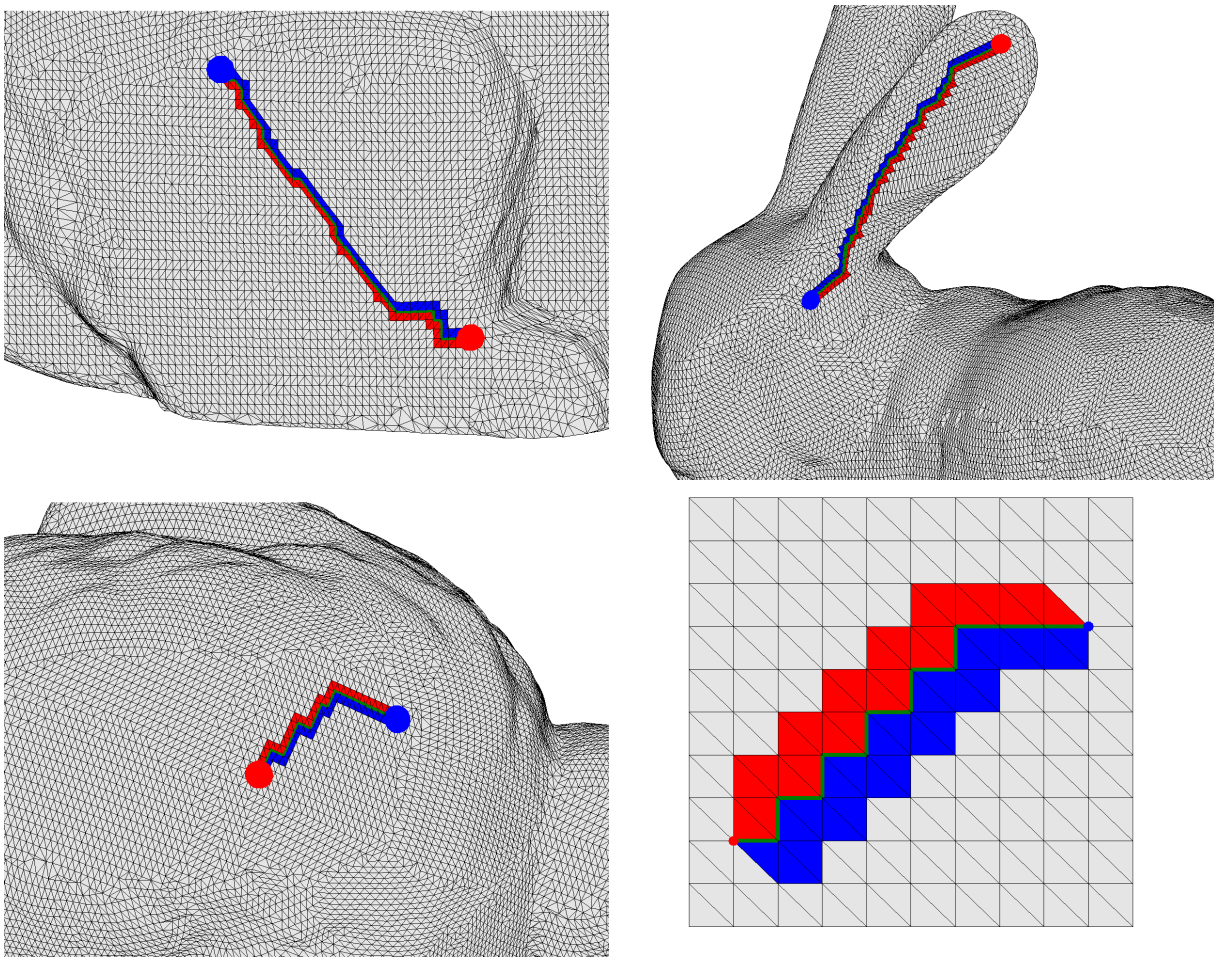

FIGURE 4 – Exemple de prise de mesure sur divers maillages triangulaires. En haut : le chemin est assez droit. La mesure sera comparable à une mesure sur l'objet réel. En bas : le chemin zigzague pour aller d'un point à l'autre. La mesure sera totalement fausse.