Etude Préalable

Projet Tutoré 2023

SUJET : Application Web 3D **TUTEUR :** DUPONT Laurent

GROUPE : LAURY Lucie, FAURE Solène, RENARD Tanguy, ZHANG Xin (RA-IL 2)

Table des matières :

I. Présentation du projet

Ce projet vise à réaliser une application web permettant de visualiser des objets 3D à partir de fichiers STL déjà existants, notamment dans le but de les transformer avant impression.
L'objectif premier pour cette application est de s'assurer de la bonne cohérence de la représentation du maillage afin que l'objet puisse être imprimé par une imprimante 3D.
Ainsi, le maillage lu dans un fichier STL sera analysé et l'application doit permettre à l'utilisateur de repérer les incohérences (trous dans le maillage, face mal orientée, face traversée par d'autres, …) et de manipuler facilement les propriétés du maillage.
Une fois toutes les opérations réalisées, l'application doit pouvoir sauvegarder ses données de maillage dans une représentation STL afin qu'il puisse être exploité facilement par les logiciels de slicing (tranchage) des imprimantes 3D.

Après avoir effectué une étude de l'existant au préalable, nous avons convenu que les principales technologies qui seront utilisées lors de ce projet sont three.js & webGL associées à une structure de données Half-Edge.

II. Fonctionnalités du système

- Visualiser un objet 3D décrit dans un fichier STL via l'application

Sur la page d'accueil de l'application, un bouton permet d'ouvrir un explorateur de fichier et sélectionner un fichier STL qu'on souhaite charger dans le navigateur. La page se fractionne alors en deux sections : l'une pour visualiser le fichier sélectionner en objet 3D et une autre correspondant au menu de modification de l'objet 3D.

- Mettre en avant un triangle du maillage sélectionné.

Lorsque l'utilisateur fait un clique gauche de la souris sur l'objet 3D, le triangle du maillage correspondant doit :

- s'afficher si le maillage n'est pas déjà visible (texture pleine de l'élément) , par un changement de couleur des arêtes, des points et de la surface,
- mettre en évidence (par un changement de couleur) les arrêtes et les point si le maillage est affiché.

- Réparer le maillage de l'objet décrit dans le fichier STL (trous, intersection de triangle, faces mal orientée)

Au chargement du fichier, le programme va vérifier la cohérence des données. Si ces dernières semblent comporter des erreurs, l'interface affiche un pop-up pour informer l'utilisateur et lui demander s'il souhaite réparer les données ou non.
Si la réponse est positive, le programme devra mettre en avant les erreurs et devra permettre à l'utilisateur de les corriger. Si la réponse est négative, l'objet est affiché tel quel.

Les problèmes pouvant être automatiquement détectés seront :

- Les triangles sans voisins. La structure doit être fermée.
- Les intersections de surfaces/ triangles pouvant gêner l'impression,
- Les faces mal orientées,
- Les points sans arêtes
- les arêtes n'appartenant qu'à un seul triangle (hors bord du modèle 3D).

- Changer la "texture" de l'objet (effet de quadrillage / lignes qui se croisent pour former l'objet).

L'objet peut être affiché en "texture pleine" mais pour certaines raisons -comme un besoin de précision- l'utilisateur peut préférer afficher le maillage de l'objet et non pas la texture. On devrait donc passer en représentation "fil de fer" qui permet de visualiser les arêtes de la structure.

Pour changer ce visuel, l'utilisateur devra cocher ou décocher une case du menu de modification. Une autre case peut également permettre d'afficher les points (intersections des arêtes) afin de les mettre en évidence.

- Pouvoir générer un nouveau point ainsi que 3 triangles autour de ce même point

Si l'utilisateur sélectionne une triangle et clique dessus, il peut choisir de rajouter un point. La surface du triangle va alors disparaître pour être remplacée par trois nouveaux triangles (donc chacun partagera une arête de l'ancien triangle).

- Changer les coordonnées d'un point

L'utilisateur peut sélectionner un point en cliquant dessus. Cette fonctionnalité est possible en vue "fil de fer" et "affichage des points". Le menu va alors afficher les coordonnées de ce point dans des input permettant de les modifier. Ou, si l'utilisateur fait un clique droit prolongé, il pourra modifier la position du point dans le plan et les coordonnées devront automatiquement être mises à jour. Les arêtes reliées à ce point doivent suivre le mouvement.

- Possibilité de pouvoir modifier l'orientation de l'objet 3D (pivot)

Certains objets 3D ont des coordonnées de points engendrant une certaine orientation globale. Seulement cette orientation n'est pas toujours naturelle par rapport au plan de la scène. De plus, les utilisateurs peuvent avoir besoin de modifier cette orientation pour imprimer d'une certaine manière l'objet.
De ce fait, le programme devrait permettre de modifier cette orientation, de faire pivoter facilement l'objet. Ceci peut se faire dans un premier temps sur des angles de 90 degré en haut, en bas, à droite, ou à gauche. Et, pour plus de précision, l'utilisateur devrait pouvoir modifier la valeur de l'angle de pivot.

Il ne s'agit pas d'une modification de la place de la caméra dans le plan ! Mais bien de la modification de toutes les coordonnées des points de l'objet 3D pour aboutir à une modification de l'orientation globale, sans déformer l'objet.

Pour ce faire, si aucun élément de l'objet 3D n'est sélectionné, le menu doit proposer un onglet "orientation" avec la valeur de l'angle et des flèches haut/bas/droite/gauche. L'action d'un clique droit de souris sur l'un de ses flèches doit faire pivoter l'objet.

Fonctionnalités éventuelles:

- Possibilité de visualiser l'objet 3D dans un environnement virtuel (utilisation de casques VR)

III. Cas d'utilisation

cas réparerFichierSTL:

Précondition le fichier STL fournit contient des problèmes de structure

Postcondition Le fichier STL est réparé, utilisable par une imprimante 3D

Déroulement normal

(1) L'utilisateur importe son fichier

(2) Le système analyse le fichier

(3) Le système détecte un ou plusieurs problème(s) de structure (trous dans le maillage, faces mal orientées, arêtes isolées, …) dans le fichier STL

(4) Le système affiche un pop-up à l'utilisateur lui indiquant que des problèmes ont été trouvés dans la structure du maillage et lui demande s'il veut voir les problèmes

(5) L'utilisateur clique sur "oui"

(6) Le système répare le fichier à l'aide des différents algorithmes qu'il possède, en suivant les actions de l'utilisateur.

(7) Le système vérifie que la structure du STL ne comporte plus de problèmes

Variantes

(A) Le fichier STL est bien structuré,

(3) Le système ne détecte pas d'erreur, le système informe l'utilisateur que son fichier ne comporte aucun problème, le cas d'utilisation se termine

(B) Le système ne parvient pas à réparer entièrement le fichier,

(8) Le système informe l'utilisateur que le fichier n'a pas pu être entièrement réparé, le cas d'utilisation se termine

Contraintes non fonctionnelles

(A) Le système doit réagir rapidement ou à minima informer l'utilisateur de l'action en cours

cas sélectionnerElémentsMaillage:

Précondition : le fichier STL fournit est importé

Postcondition : les éléments sélectionnés s'affichent d'une autre couleur

Déroulement normal :

(1) l'utilisateur clique sur un élément (point ou arête) du maillage

(2) le système change de couleur l'élément sélectionné.

Variante :

(A) Un élément est déjà sélectionné et l'utilisateur clique sur un autre élément sans presser la touche "CTRL".

→ le système ne considère plus l'élément déjà sélectionné et leur fait perdre leur couleur.

→ l'élément sélectionné apparaît en orange.

(B) Un élément est déjà sélectionné et l'utilisateur clique sur un autre élément en pressant la touche "CTRL".

→ le système met en couleur le nouvel élément sélectionné sans désélectionner le premier élément : sélection multiple

(C) L'utilisateur clique sur une face.

→ le système doit mettre en couleur orange les arêtes et points de cette face.

(D)l'utilisateur clique en dehors de la figure : aucun élément doit être sélectionné

cas changerTexture

Précondition un fichier STL est importé et ne contient pas de problème de structure

Postcondition La "texture" du modèle 3D sur l'application est changé

Déroulement normal

(1) L'utilisateur accède au menu de modification

(2) L'utilisateur coche la case "Fil de fer" dans l'onglet affichage

(3) Le système transforme la texture du modèle 3D en une version "fil de fer"

(4) Le système affiche la nouvelle version du modèle 3D

Variantes

(A) L'utilisateur décoche la case de "fil de fer", (3) Le système remet la texture du modèle 3D à son état d'origine, (4) Le système affiche le modèle avec sa texture d'origine

Contraintes non fonctionnelles

(A) Le système doit réagir rapidement ou à minima informer l'utilisateur de l'action en cours

cas ModifierCoordonnéesPoint

Précondition un fichier STL est importé et ne contient pas de problèmes de structure
Vue “Fil de fer” & “Affichage des points”.

Postcondition La position du point en question est changée, et les arêtes connectées
doivent suivre ce mouvement.

Déroulement normal

(1) L’utilisateur clique sur un point

(2) L’utilisateur accède à un menu affichant les coordonnées de ce point et permettant de les modifier.

(3) L’utilisateur modifie les coordonnées

(4) Le système affiche la nouvelle version du modèle 3D avec le point à ses nouvelles coordonnées

Variantes

(A) L’utilisateur effectue un clic droit prolongé,
(2) L’utilisateur peut faire glisser le point à l’endroit souhaité
(3) Le système doit mettre à jour les coordonnées du point correspondant à sa nouvelle position.

cas ModifierCoordonnéesPoint

Précondition un fichier STL est importé et ne contient pas de problèmes de structure

Postcondition: L'orientation du plan est changée.

Déroulement normal

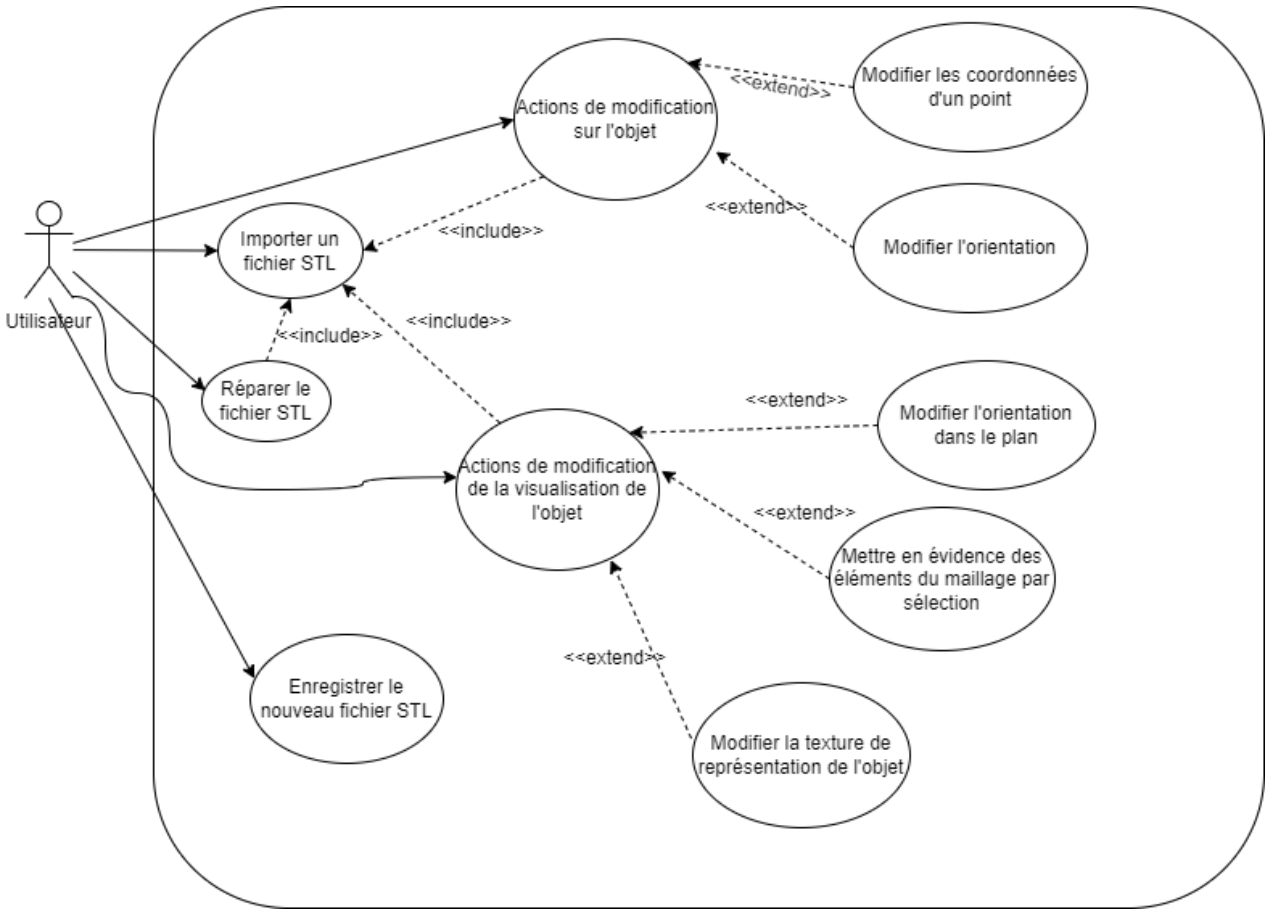
(1) L’utilisateur accède au menu “Modification”

(2) L’utilisateur clique sur le bouton “up”, “down”, “left” ou “right”

(3) Le système modifie l’orientation du modèle 3D sans le déformer

(4) Le système affiche le modèle 3D nouvellement orienté

Diagramme des cas d'utilisations:



Scénario 1 :

Victor importe son fichier exemple.stl
Le système accepte le fichier et vérifie s'il y a des problèmes dans le maillage
Le système détecte des faces mal orientées et un trou dans le maillage
Le système informe Victor qu'il y a des problèmes et lui propose de les visualiser
Victor clic sur "Oui"
Le système met en évidence les éléments posant problèmes
Victor effectue les modifications nécessaires
Le système ne détecte plus de problème
Victor ouvre le menu de modification et clic sur la case "Fil de fer"
Le système transforme la texture du modèle 3D en "fil de fer"
Victor enregistre son nouveau modèle 3D au format STL

IV. Recensement et évaluation des risques

Le plus gros risque réside dans la structure de données. Il est important de bien choisir la manière dont les données seront structurées après la lecture du fichier STL. En effet, ces derniers peuvent contenir une quantités importante de données c'est pourquoi notre structure doit être efficace et optimisée.

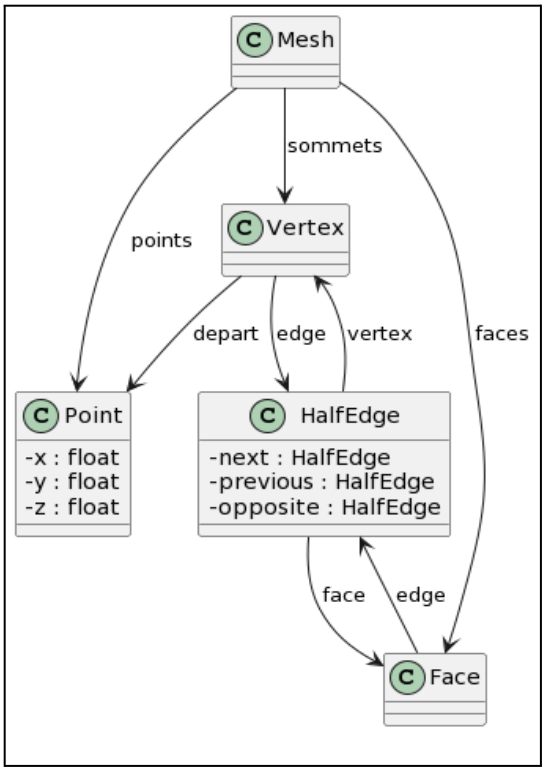
De plus, la structure devra permettre de vérifier les différents éléments pour s'assurer de la bonne composition du maillage. Les données doivent donc être organisées afin d'accéder rapidement aux informations. De ce fait, la structure doit être dès le départ bien pensée afin de pouvoir y appliquer tous les algorithmes dont on aura besoin. Partir sur une structure de données mal adaptées pourrait nous amener à devoir la redéfinir plus tard lors du développement et risquer de perdre beaucoup de temps à refaire les algorithmes déjà en place.

Au vu des modifications qui pourraient s'appliquer à chaque point, qui seront stockés dans différentes arêtes, il est important de veiller à ce qu'un point soit représenté par une seule et unique instance (car un point sera stocké dans plusieurs arêtes) afin d'assurer une modification globale.

Pour finir, la structure de données doit permettre l'écriture d'un nouveau fichier STL pour enregistrer les modifications.

V. Structure de données

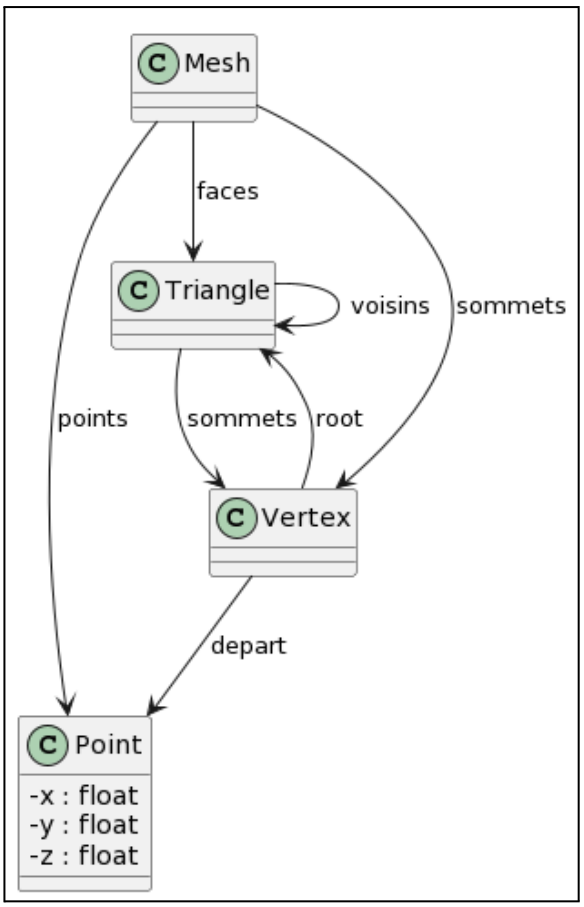
Nous allons utiliser une structure de données de type "Half-Edge" (ou demie-arête en français).



Cette structure a 2 représentation possibles :

- 1ère représentation : basée sur des polygones
    - point (3 coordonnées),
    - sommets (point p + sa demie-arête),
    - faces composée d'une de ses demie-arêtes bordière permettant ensuite de tourner autour de cette face en utilisant ses autres demies-arêtes).
    - Demie-arête définie par un sommet de départ, la face à laquelle elle appartient, ainsi qu'une référence vers les demies-arêtes précédente, suivante et opposée (la même mais dans le sens inverse).

- 2ème représentation : basée sur des maillage triangulaire
    - Point : 3 coordonnées
    - Sommet : Point + Triangle auquel appartient le sommet
    - Un sommet + les 3 triangles voisins



On fera le choix d'implémenter la première structure de données

VI. Prévision d'algorithmes

Il est à présent important de comprendre comment va s'effectuer le remplissage de ses données à la lecture d'un fichier STL.

```
algorithme remplissageDesDonnes
fonction convertirSTLtoDonnees (stl : Fichier) : Mesh
début

points ← Tableau[0..n]
vertices ← Tableau[0..n]
faces : Tableau[0..n/3]

Tantque non EOF faire

p1 ← lire_ligne(stl)
p2 ← lire_ligne(stl)
p3 ← lire_ligne(stl)

si points contient p1  alors
     p1  ← p1bis
     sinon ajouter(p1, points)
fsi
//pareil pour les deux autres points

v1 <- new Vertex(p1)
h1 <- new Halfedge(v1)
v1.halfedge <- h1
v2 <- new Vertex(p2)
h2 <- new Halfedge(v2)
v2.halfedge <- h2
 v3 <- new Vertex(p3)
h3 <- new Halfedge(v3)
v3.halfedge <- h3

h1.next <- h2
h2.previous <- h1
h2.next <- h3
h3.previous <- h2
h3.next <- h1
h1.previous <- h3
nouvelleFace <- new Face(h1)
h1.face <- nouvelleFace
h2.face <- nouvelleFace
h3.face <- nouvelleFace
```

```
ajouter(faces, nouvelleFace)

//détection des arêtes opposées pour compléter la structure de données
vertexPoint1 <- vertices.filter( element → element.point == p1 )
si vertexPoint1 contient des valeurs
      alors
            halfedgeOppose <- vertexPoint1.map(element -> element.halfedge.previous)
                  .filtrer(element -> element.vertex.point == p2)
                  .findFirst()
            si halfedgeOppose non null alors
                  si halfedgeOppose.opposite est null alors
                        halfedgeOppose.opposite <- h1
                        h1.opposite <- halfedgeOppose.opposite
                       sinon erreur
                 finsi
            finsi
finsi
//pareil pour les deux autres points

ajouter(vertices, v1)
ajouter(vertices, v2)
ajouter(vertices, v3)
ftant

points ← trierPoints(points)
//Trier vertex par points
//Trier face par vertex

mesh ← Mesh(faces, vertices, points)
retourne mesh

fin
```

Lexique

- stl : Fichier Texte, le fichier STL à convertir
- points : Tableau de N Point, tableau de tous les points du maillage
- vertices : Tableau de N Vertex, tableau de tous les vertex du maillage
- Point = <x: entier, y: entier, z: entier>, position d'un point dans le plan
- Vertex = <point : Point, halfedge : Halfedge>, représentation d'un sommet dans le plan
- Halfedge = <next : Halfedge, previous : Halfedge, opposite : Halfedge, vertex : Vertex, face : Face> représentation d'une arête dans le plan
- vertexPoint1 : Tableau de N Vertex, les vertex ayant pour point de départ le point p1.
- halfedgeOppose: Halfedge, l'arête en face de celle en construction, soit son opposée

```
fonction trierPoints(points : Tableau[0..n]) : Tableau
début
pour i de 1 à n-1 faire
        pointCourant ← points[i]
        j ← i - 1
        Tant que j >= 0 et comparerPoints(pointCourant, points[j]) < 0 faire
                points[j+1] ← points[j]
                j ← j - 1
        ftant
        points[j+1] ← pointCourant
fpour

retourne points
fin
```

```
fonction comparerPoints(pointA : Point, pointB : Point) : entier
début

    si pointA.x ≠ pointB.x alors
            retourne pointA.x - pointB.x

    sinon si pointA.y ≠ pointB.y alors
            retourne pointA.y - pointB.y

    sinon si pointA.z ≠ pointB.z alors
            retourne pointA.z - pointB.z

fin
```

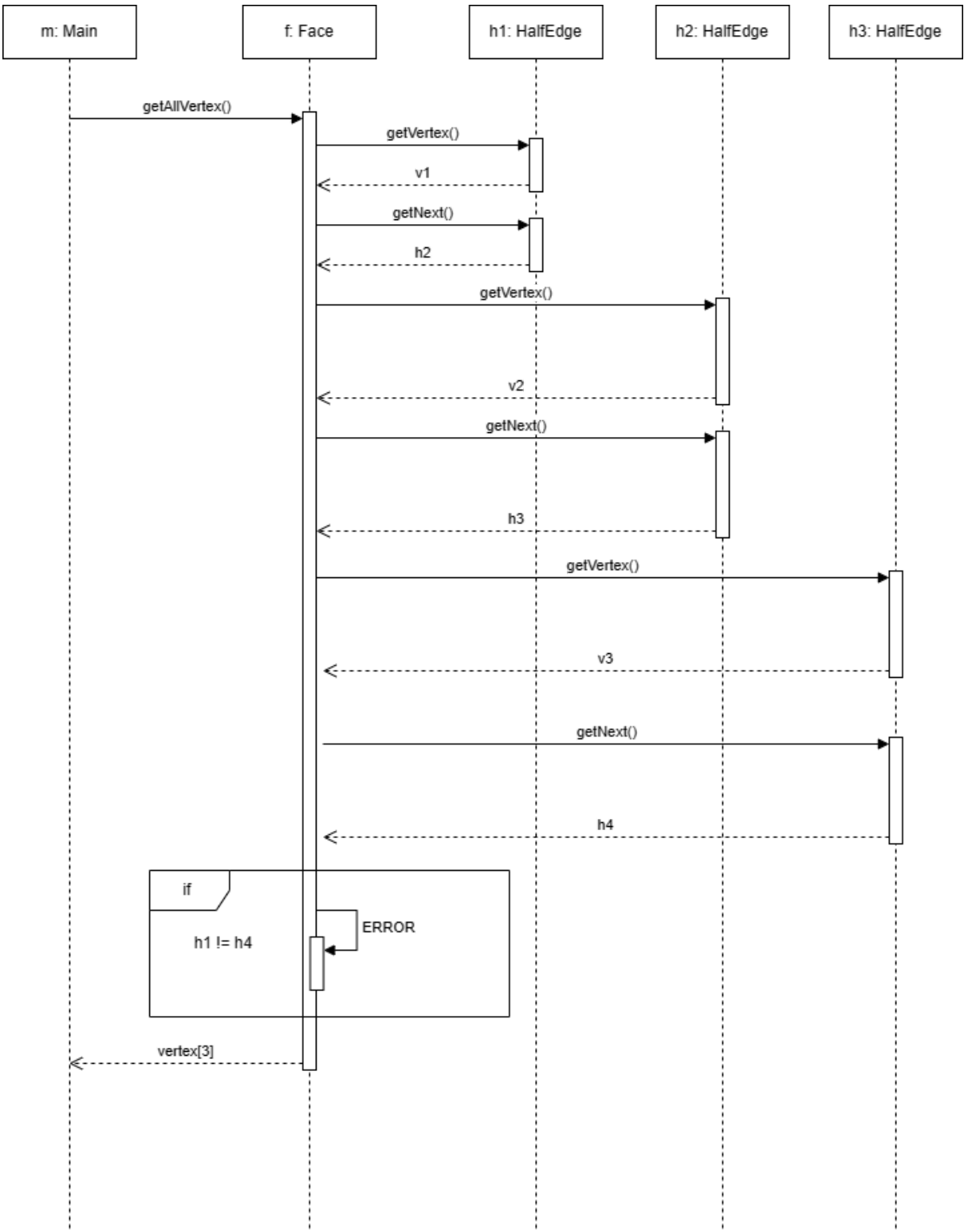
Actions sur les arêtes d'une face :
On peut compter le nombre d'arêtes d'une face grâce à cet algorithme :

```
public int degree() {
    Halfedge<X> e,p;
    if(this.halfedge==null) return 0;

    e=halfedge; p=halfedge.next;
    int cont=1;
    while(p!=e) {
        cont++;
        p=p.next;
    }
    return cont;
}
```

Méthode retrouvée dans Mesh representations and data structures de Lucas Aleardi (voir sources)

Dans cette méthode on récupère les données d'une face (Halfedge). Pour cette donnée de départ, on va prendre l'halfedge suivante, ce qui représente une arête pour une face, jusqu'à ce qu'on revienne à la face de départ. On compte à chaque appel du prochain halfedge pour savoir combien il y en a. Au début du développement de l'application, on considère que le maillage doit être représenté par des triangles. Donc si le nombre de halfedge n'est pas à 3, on considérera qu'il y a un problème dans le maillage (soit qu'il manque des arêtes (count > 3), soit que la face n'est pas fermée (count < 3)).

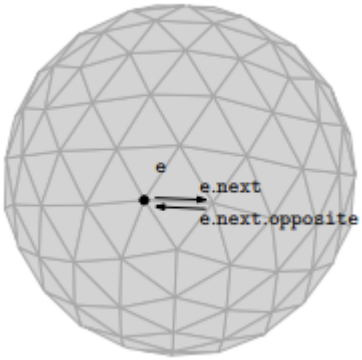
Pour une face, on peut récupérer les trois arêtes correspondantes :

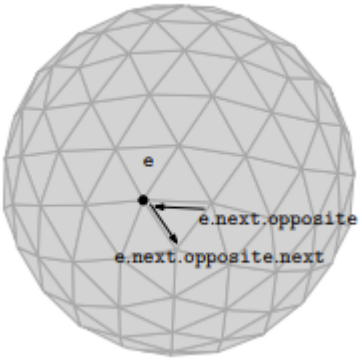

m: Main
f: Face
h1: HalfEdge
h2: HalfEdge
h3: HalfEdge
getAllVertex()
getVertex()
v1
getNext()
h2
getVertex()
v2
getNext()
h3
getVertex()
v3
getNext()
h4
if
h1 != h4
ERROR
vertex[3]

Action sur les faces adjacentes :

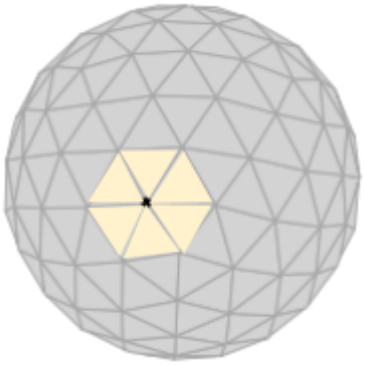
On peut calculer le nombre de faces adjacentes ayant un point commun:

```
public int vertexDegree(Vertex<X> v) {
    int result=0;
    Halfedge<X> e=v.getHalfedge();

    Halfedge<X> pEdge=e.getNext().getOpposite();
    while(pEdge!=e) {
        pEdge=pEdge.getNext().getOpposite();
        result++;
    }

    return result+1;
}
```



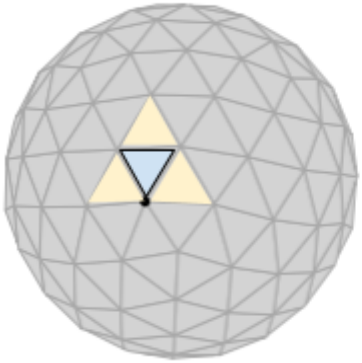


Dans cette méthode, on compte le nombre d'arêtes récupérées autour d'un point en partant d'une arête d'une face.
"e" représente une halfedge (arête). e.next est l'arête suivante d'une même face. e.next.opposite représente cette même arête mais qui a pour point de départ le point d'arrivé de e.next. C'est une arête rattachée à la face adjacente à la face sur laquelle on s'est positionné au départ. On a donc basculé sur la face adjacente, et en répétant l'opération on va parcourir les faces ayant toutes un point commun (point noir sur le schéma).



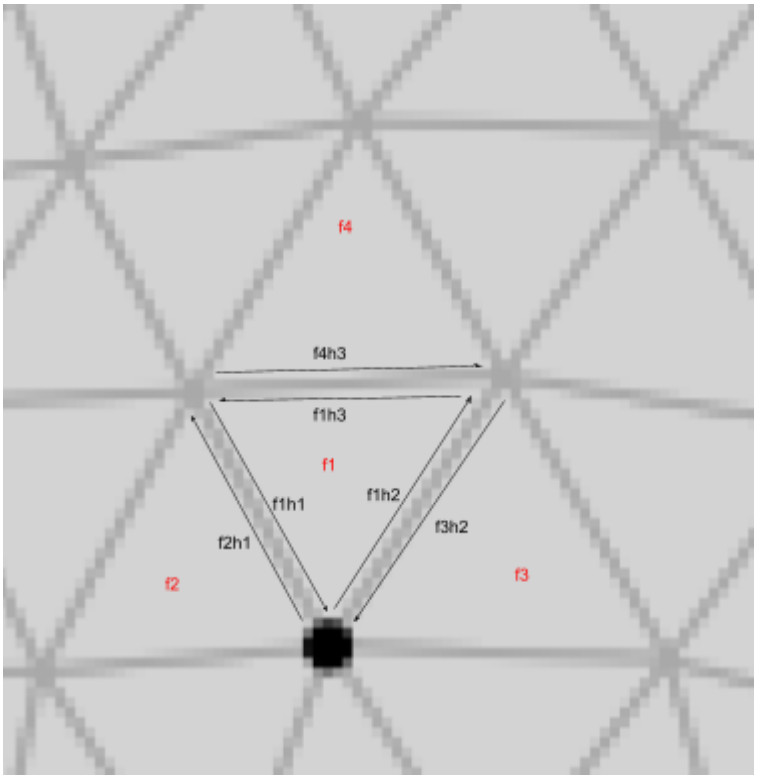
Dans cette méthode on ne fait que les compter, mais on peut très bien les récupérer.

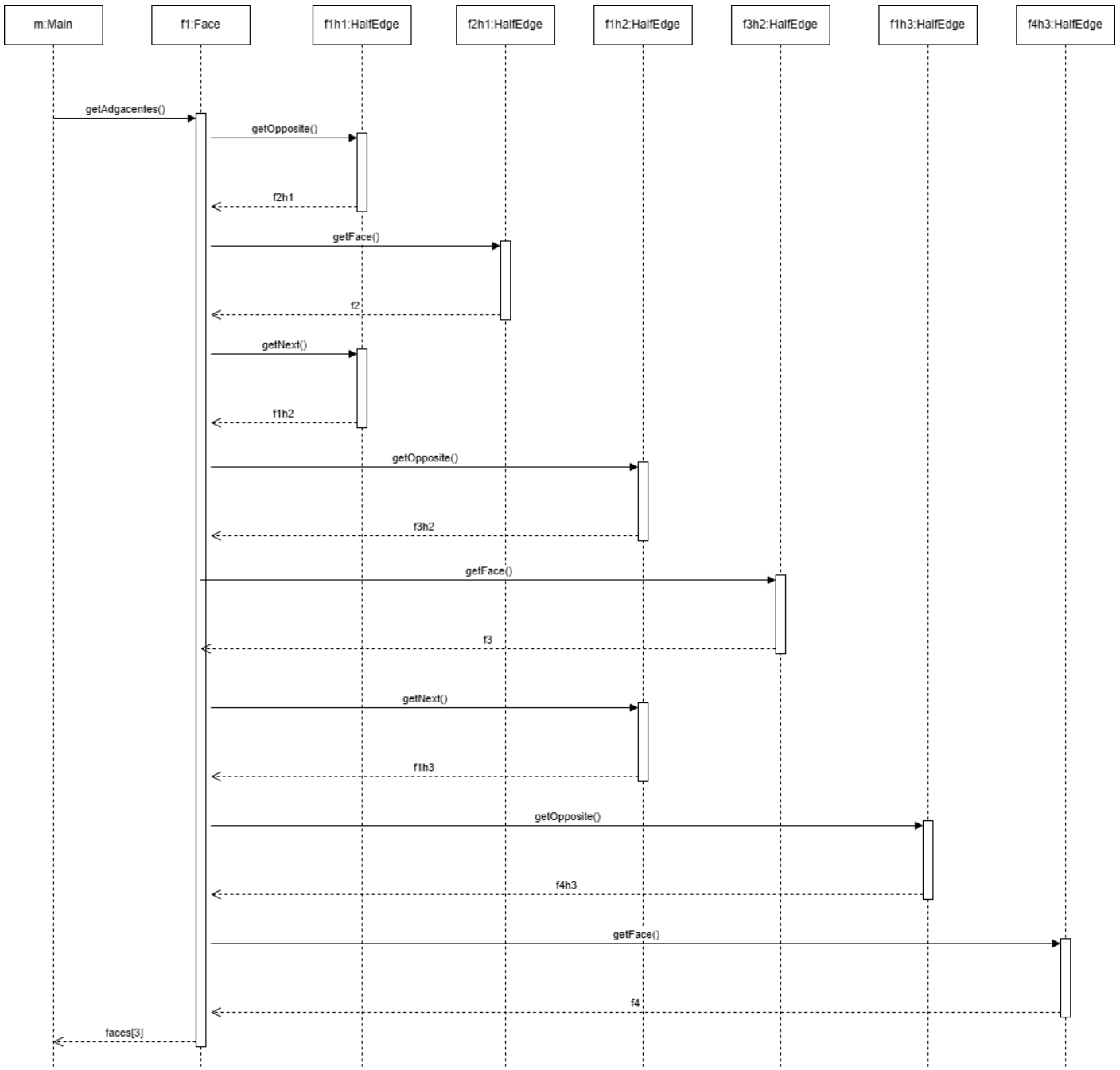
Pour un point d'une face, on peut également récupérer les trois faces adjacentes à ses arêtes :



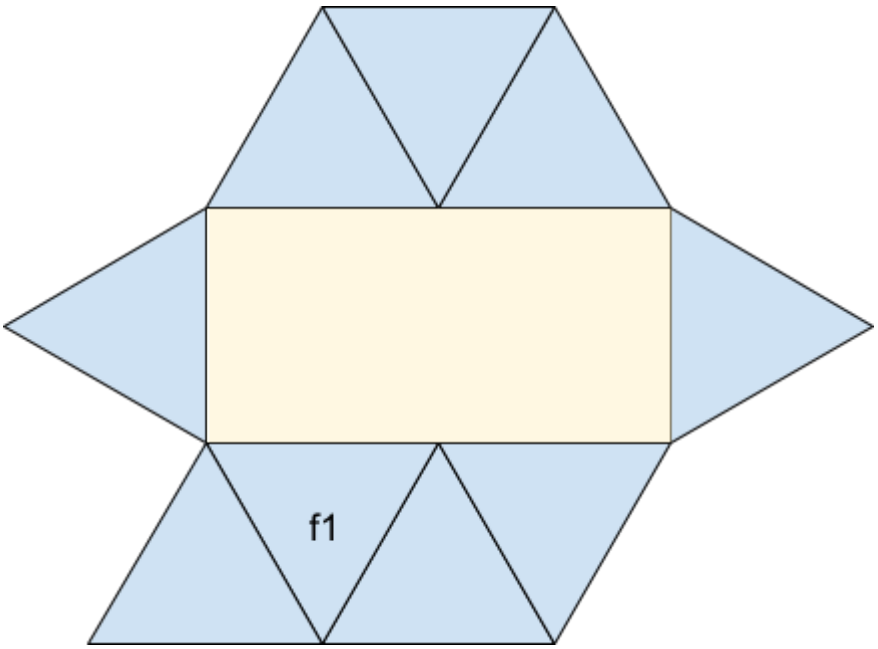
= faces dont on veut connaître les faces adjacentes

= faces adjacentes

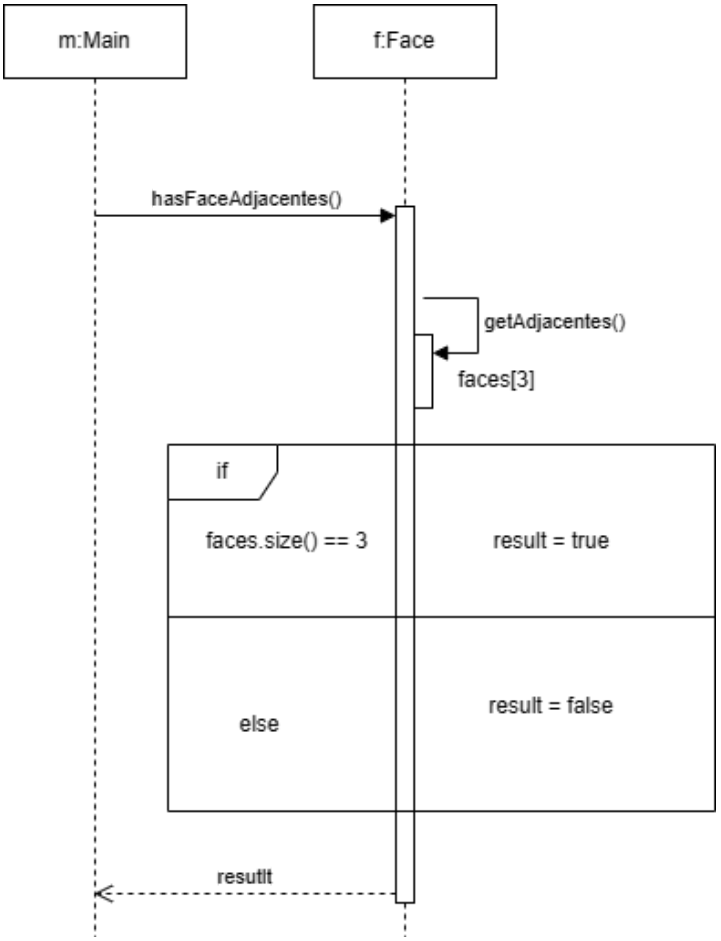




Grâce à cette méthode, on peut détecter s'il y a un "trou" dans le maillage. C'est-à-dire si la face analysée a bien trois faces adjacentes. Si elle ne les a pas, c'est que le maillage est mal formé.
exemple de trou :

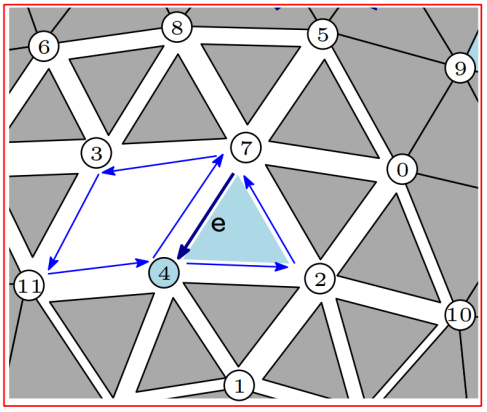


si on analyse la face f1, la méthode de vérification des cases adjacentes relèvera l'anomalie : il lui manque une face adjacente et donc il y a un trou dans le maillage



- **Vérifier que toutes les normales des faces sont bien orientées :**

La structure half-edge donne un sens de considération des faces anti-horaire :



Ainsi, pour vérifier qu'une face est bien orientée, il faut vérifier que son sens des arêtes est bien anti-horaire. Pour cela, on calcule le produit vectoriel de ses 3 arêtes et on vérifie qu'il est bien supérieur à 0 :

```
fonction estAntiHoraire(face : Face) : Booléen
début

        h1 ← face.getHafEdge()
        v1 ← h1.getVertex()
        p1 ← v1.getPosition()

        h2 ← h1.next()
        v2 ← h2.getVertex()
        p2 ← v2.getPosition()

        h3 ← h2.next()
        v3 ← h3.getVertex()
        p3 ← v3.getPosition()

        antiHoraire ← faux

        produitVectorielX = ( (p2.getY() - p1.getY()) * (p3.getZ() - p1.getZ()) ) - ((p2.getZ() -
p1.getZ()) * (p3.getY() - p1.getY()) )
        produitVectorielY = ( (p2.getZ() - p1.getZ()) * (p3.getX() - p1.getX()) ) - ((p2.getX() -
p1.getX()) * p3.getZ() - p1.getZ()))
        produitVectorielZ = ( (p2.getX() - p1.getX()) * (p3.getY() - p1.getY()) ) - ((p2.getY() -
p1.getY()) * p3.getX() - p1.getX()))

         produitVectoriel = produitvectorielX + produitVectorielY + produitVectorielZ
```

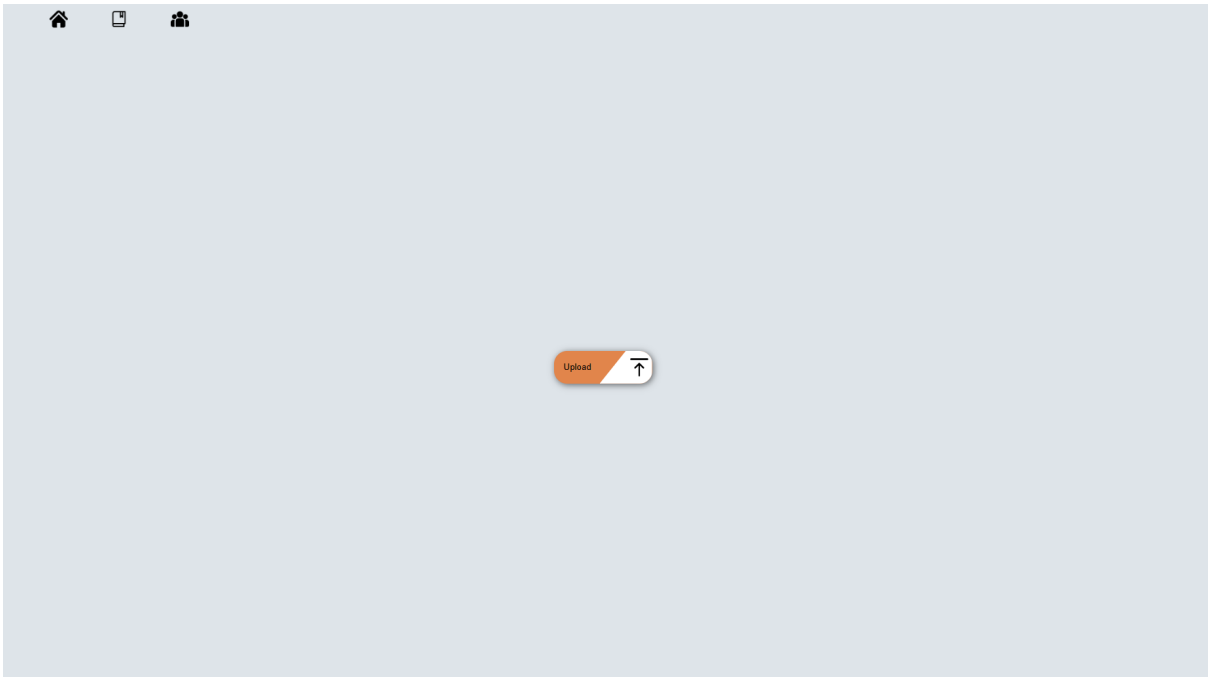
```
        si produitVectoriel > 0 alors
            antihoraire ← vrai
        sinon antihoraire ← faux
        fsi
        retourne antihoraire
fin
```

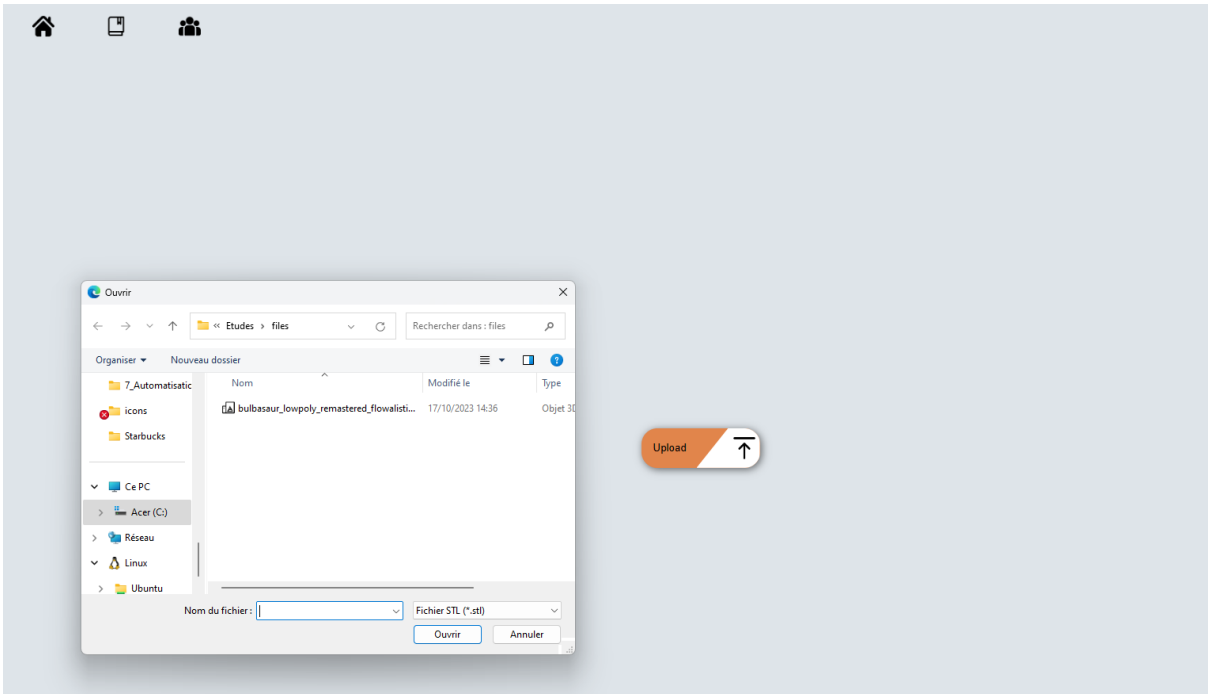
Lexique :

- h1 : Half-Egde, arête d'une face
- h2 : Half-Egde, arête d'une face
- h3 : Half-Egde, arête d'une face
- v1 : Vertex, sommet de départ de l'arête h1
- v2 : Vertex, sommet de départ de l'arête h2
- v3 : Vertex, sommet de départ de l'arête h3
- p1 : Point, point correspondant au sommet v1
- p2 : Point, point correspondant au sommet v1
- p3 : Point, point correspondant au sommet v1
- Point = <x: entier, y: entier, z: entier>, position d'un point dans le plan
- Vertex = <point : Point, halfedge : Halfedge>, représentation d'un sommet dans le plan
- Halfedge = <next : Halfedge, previous : Halfedge, opposite : Halfedge, vertex : Vertex, face : Face> représentation d'une arête dans le plan
- produitVectorielX : entier, produit vectoriel des coordonnées X des 3 points des sommets
- produitVectorielY : entier, produit vectoriel des coordonnées Y des 3 points des sommets
- produitVectorielZ : entier, produit vectoriel des coordonnées Z des 3 points des sommets
- produitVectoriel : entier, somme des produits vectoriel donnant l'orientation de la normale de la face

VII. Maquette de l'application

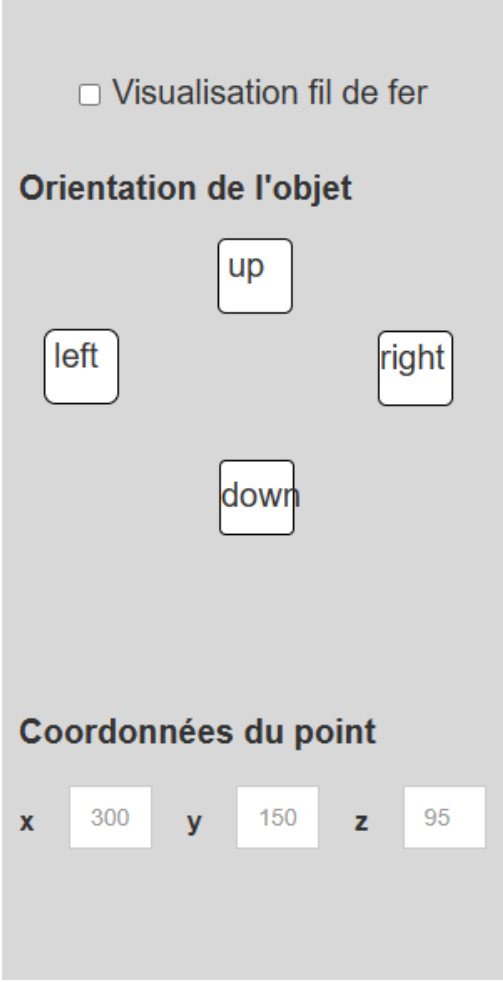
- écran de choix du fichier





- écran d’affichage du fichier

- Menu de modification



VIII. Planning des itérations

I. **Itération 1 (42h)**
- importer un fichier STL et visualiser un objet 3D
- implémentation de la structure des données
    - faire le tri des points à la lecture du fichier stl (ordre lexicaux graphique) et reconnaître les points qui sont égaux ou non (suivant la lecture des coordonnées)
    - ordre lexicaux graphique : comparer les x, puis les y, puis les z
    (si x1 > x2 alors point1 > point2, si égaux, on regarde de la même manière les y)
    - trier les points dans l'ordre pour faciliter les algos derrière
- implémentation de l'algorithme de parcours de la structure et afficher à l'écran
- implémentation de l'algorithme de détection des trous
- implémentation de la fonctionnalité de mise en avant d'un triangle du maillage
- commencer l'implémentation de l'algorithme d'actions sur les arêtes d'une face
- commencer l'implémentation de l'algorithme d'actions sur les faces adjacentes

II. **Itération 2 (28h)**
- finir l'implémentation des algorithmes d'actions sur les arêtes et les faces
- implémentation de la fonctionnalité de changement de "texture" de l'objet
- implémentation de la fonctionnalité de changement des coordonnées d'un point
- implémentation de la fonctionnalité de génération d'un nouveau point ainsi que 3 triangles autour de ce même point

III. **Itération 3 (28h)**
- implémentation de la fonctionnalité de modification de l'orientation de l'objet (modification des coordonnées)
- commencer l'implémentation de la fonctionnalité de réparation du maillage de l'objet du fichier STL

IV. **Itération 4 (28h)**
- Préparation de la soutenance de fin de semestre
- finaliser l'implémentation de la fonctionnalité de réparation du maillage de l'objet du fichier STL
- si possible, aborder la fonctionnalité de visualisation d'un objet 3D dans un environnement virtuel avec des casques VR.

V. **Itération 5**
- Préparation de la soutenance finale
- Finaliser l'implémentation des fonctionnalitées
- si possible, aborder la fonctionnalité de visualisation d'un objet 3D dans un environnement virtuel avec des casques VR.

VI. **Itération 6**
- Finalisation de la soutenance finale
- Rectification de potentiels problèmes rencontrés

IX. Sources / Inspirations

- Polytechnique | Mesh representations and Data structures - Lucas Aleardi
- OpenMesh - a generic and efficient polygon mesh data structure
- Interactive Explainers for Geometry Processing Algorithms
- THE HALF-EDGE DATA STRUCTURE | Dan Englesson