



IUT Nancy-Charlemagne
Université de Lorraine
2 ter Boulevard Charlemagne
BP 55227
54052 Nancy Cedex

Département Informatique

Création d'une application web à des fins d'impressions 3D

Rapport de fin de semestre du projet tutoré

Groupe : LAURY Lucie, FAURE Solène, ZHANG Xin, RENARD Tanguy

Tuteur: DUPONT Laurent

Année scolaire 2023-2024

Remerciements :

Nous tenions à remercier notre tuteur pour ce projet, M. DUPONT Laurent, qui a pris de son temps pour nous aider quand nous avons un problème, et nous aiguiller, tant à l'IUT qu'à distance. En espérant que sa disponibilité puisse se poursuivre jusqu'à la fin de ce projet.

Nous remercions également Mme. DEBLED-RENNESSON Isabelle pour son suivi, ses remarques constructives tout au long de l'avancée de ce projet et ses propositions d'améliorations. Son point de vue externe a été un élément de progression quant à certains problèmes que nous avons rencontrés.

Table des matières :

Introduction.....	4
Partie 1 : Analyse.....	5
I. Découpage fonctionnel du projet.....	5
II. Evolution par rapport à l'étude préalable de décembre.....	6
1. Différence pour l'expérience utilisateur.....	7
2. Algorithme de remplissage des données.....	10
A. Présentation.....	10
B. Calcul de complexité d'algorithmes de gestions de listes.....	16
C. Calcul de complexité d'algorithme de remplissage de données.....	19
Partie 2 : Réalisation du projet.....	26
I. Architecture logicielle.....	26
1. Structure HalfEdge.....	26
2. SkipList.....	27
II. Difficultés rencontrées.....	30
III. Déroulement du projet.....	31
1. Planning du projet.....	31
2. Répartition des tâches.....	32
IV. Présentation d'un élément original dont chaque étudiant est fier.....	33
V. Objectifs pour la fin du projet.....	36
VI. Planning pour les deux itérations restantes.....	37
Conclusion.....	38

Introduction

Lors du projet tutoré, organisé durant la troisième année du Bachelor Universitaire de Technologie, les étudiants en informatique doivent travailler en équipe de quatre ou cinq afin de réaliser un projet. Pour notre part, notre objectif est de concevoir une application web de visualisation et modélisation d'un objet 3D à des fins d'imprimerie.

Après une phase d'analyse et d'étude préalable du projet, nous avons pu nous concentrer sur la réalisation de cette application durant tout le mois dernier. Ces dernières semaines ont été divisées en quatre itérations, la première étant la plus conséquente. Nous arrivons maintenant à la fin de la quatrième itération.

Le projet a pu ainsi avancer grandement durant cette période, permettant que les fonctionnalités jugées les plus importantes et nécessaires ont pu être réalisées. Les deux itérations restantes seront dédiées à améliorer le projet.

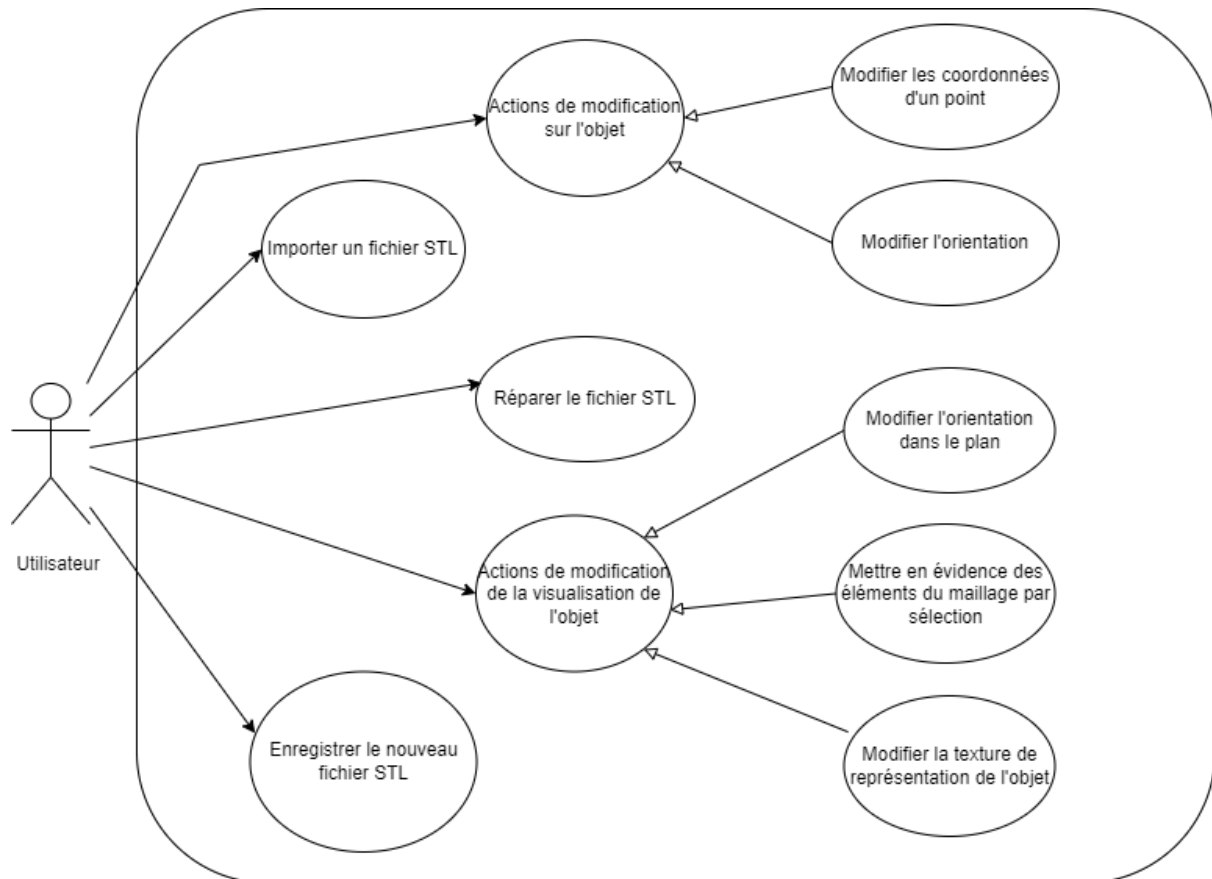
Ce projet a vu quelques changements au cours des itérations par rapport à l'étude préalable, notamment en termes de remplissage de la structure de données, de changement dans les plannings ou encore des fonctionnalités ajoutées.

Chacun des membres de l'équipe a pu apporter sa pierre à l'édifice pour en arriver à ce stade du projet.

Partie 1 : Analyse

I. Découpage fonctionnel du projet

Résumé des cas d'utilisations:



Scénario :

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

II. Evolution par rapport à l'étude préalable de décembre

Bien que la majorité du projet ait pu suivre l'idée initiale que nous avons, quelques changements ont eu lieu par rapport à l'étude préalable.

Le changement le plus important s'est avéré être l'algorithme de remplissage des données. Avec le premier algorithme que nous avons, il fallait parfois attendre plus d'une heure pour des modèles assez conséquents. Il était donc nécessaire de trouver une alternative pour améliorer grandement ce temps pour l'utilisateur. Finalement, nous avons pu réduire le temps de remplissage d'une manière conséquente (de 1h à 1-2 min pour le même modèle conséquent), notamment en utilisant des SkipList, structure de données expliquée en 2ème partie de ce rapport..

Il y a eu aussi des changements au niveau de l'expérience utilisateur, notamment au niveau du design, bien que ces changements soient négligeables.

Le planning des itérations s'est avéré être assez différent de ce qui était prévu, mais la majorité des fonctionnalités ont pu être réalisées à l'itération qui leur était assignée. Les changements de planning ont notamment concerné des fonctionnalités dont nous n'avions pas pensé lors de l'étude préalable ou qui ont pris plus de temps que prévu.

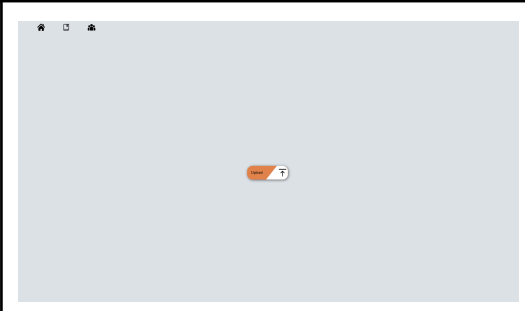
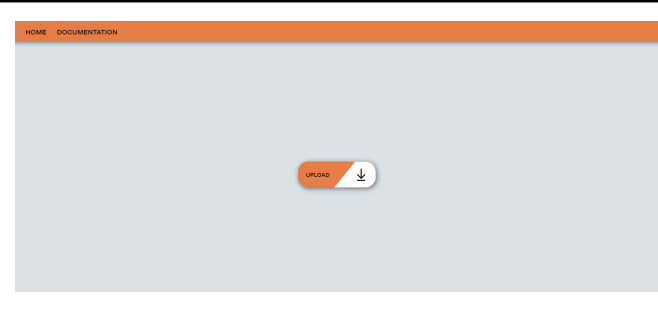
1. Différence pour l'expérience utilisateur

- **Design**

La communication entre l'application et l'utilisateur a subi quelques modifications par rapport à l'étude préalable. Dans un premier temps, le menu de modification a été amélioré afin d'inclure trois onglets différents et faciliter la recherche de l'information.

Etude Préalable	Réalisé
<div><input type="checkbox"/> Visualisation fil de fer</div> <div>Orientation de l'objet</div> <div><div>up</div><div>left</div><div>right</div><div>down</div></div> <div>Coordonnées du point</div> <div>x <input type="text" value="300"/> y <input type="text" value="150"/> z <input type="text" value="95"/></div>	<div><div>VISUALISATION</div><div>MODIFICATION</div><div>PROBLÈMES</div></div> <div>GRID <input checked="" type="checkbox"/></div> <div>MODE DE SELECTION D'UNE FACE(NOM TEMPORAIRE) <input type="checkbox"/></div> <div>TEXTURE "FIL DE FER" <input checked="" type="checkbox"/></div> <div>ANTI-ALIASING <input type="checkbox"/></div>
<div>Coordonnées du point</div> <div>x <input type="text" value="300"/> y <input type="text" value="150"/> z <input type="text" value="95"/></div>	<div><div><div>A : X <input type="text" value="-0,500"/> Y <input type="text" value="-0,500"/> Z <input type="text" value="0,500"/></div><div>B : X <input type="text" value="0,500"/> Y <input type="text" value="-0,500"/> Z <input type="text" value="0,500"/></div><div>C : X <input type="text" value="0,500"/> Y <input type="text" value="0,500"/> Z <input type="text" value="0,500"/></div></div></div>

Le header et le bouton d'import ont eux-aussi légèrement changé par rapport à la maquette initiale

Etude Préalable	Réalisé
	

• Fonctionnalités

La fonctionnalité de réparation de l'objet 3D était prévue comme étant primordiale mais n'a pas encore été réalisée. Elle était en effet prévue au début du projet mais sera finalement implémentée à la fin. Parmi les problèmes que nous pensions relevés, l'un d'entre eux est un cas que nous ne rencontrons pas : les points sans arêtes. Ainsi, nous ne considérons plus cette erreur dans la suite du développement de l'application.

Le cas d'utilisation initialement prévu pour cette fonctionnalité est également modifié :

Cas d'utilisation de l'étude préalable :

cas réparerFichierSTL:

Précondition : le fichier STL fourni 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

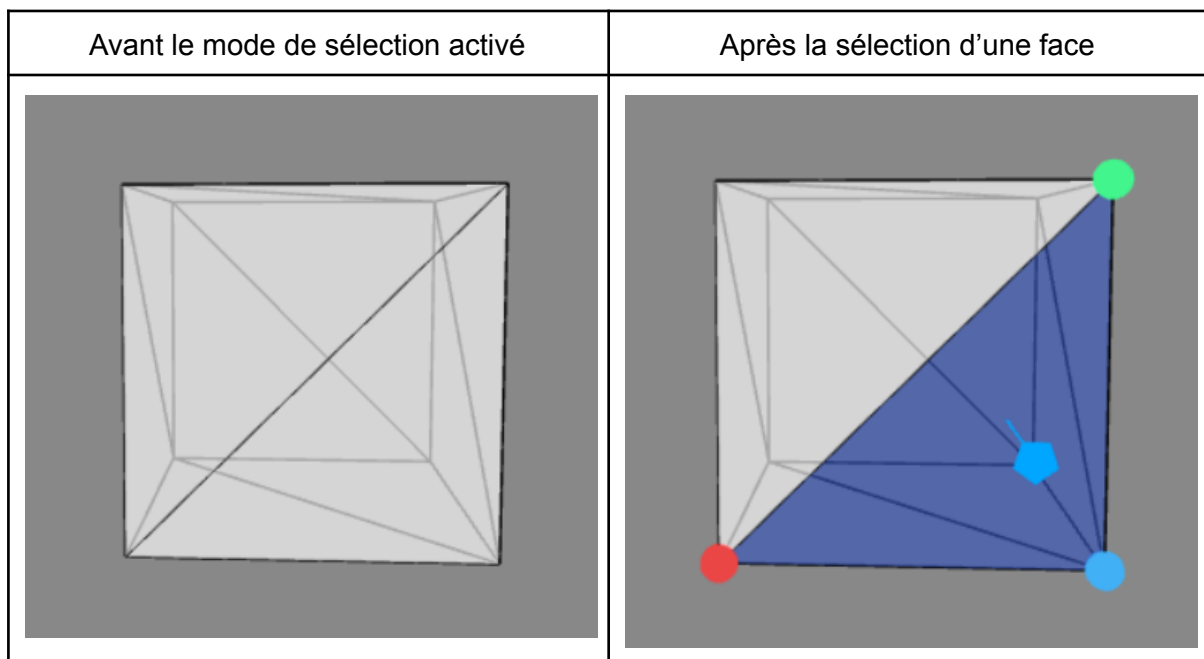
(A) Le système doit réagir rapidement ou à minima informer l'utilisateur de l'action en cours

A l'import du fichier, le système l'analyse. Or aucun pop-up n'est affiché : les erreurs sont automatiquement communiquées dans l'onglet "Problèmes" du menu de modification et sont affichées sur la structure 3D.

Il n'y a pour le moment pas de réparation de fichier possible, donc il n'y a pas non plus de bouton de validation.

La sélection des éléments du maillage se déroule également différemment. Bien que nous ayons pensé au préalable permettre cette fonctionnalité au simple clic de la souris sur l'élément désiré, nous avons finalement préféré ajouter un mode de sélection dans le menu de modification.

Cela nécessite de cocher la case "mode de sélection" pour pouvoir sélectionner une face de la structure. En cliquant sur la face désirée, elle s'affiche en bleu. Une fois la face sélectionnée, les trois sommets correspondant sont également mis en avant par des sphères de couleurs.



Les variantes considérées dans ce cas d'utilisation telles que la possibilité de sélectionner plusieurs face en maintenant la touche "CTRL" activée n'a pas été implémentée. Elle est prévue dans la suite du projet.

Pour finir, **la modification de l'orientation de l'objet global** s'effectue sans nécessiter d'action dans le menu de modification. Three.js fournit des méthodes permettant de modifier son orientation par action de la souris dans le plan avec l'enregistrement des nouvelles coordonnées de chaque point.

2. Algorithme de remplissage des données

A. Présentation

Notre diagramme de classe représentant notre structure de données a subi quelques modifications, détaillées en 2ème partie de ce rapport. Cela a permis à l'algorithme nous permettant de convertir les données du fichier STL en données de l'application d'être drastiquement amélioré. En effet, à l'issue de la première itération, le déroulement de cet algorithme conduisait à un chargement dans l'application pouvant dépasser une heure, bloquant pendant autant de temps l'utilisateur. Nous avons donc réfléchi à un nouvel algorithme plus efficace et performant.

Pour rappel, l'**ancien** algorithme :

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

1.   points ← Tableau[0..n]
2.   vertices ← Tableau[0..n]
3.   faces : Tableau[0..n/3]
4.
5. Tantque non EOF faire
6.
7. p1 ← lire_ligne(stl)
8. p2 ← lire_ligne(stl)
9. p3 ← lire_ligne(stl)
10.
11. si points contient p1 alors
12.   p1 ← p1bis
13. sinon ajouter(p1, points)
14. fsi
15. //pareil pour les deux autres points
16.
17. v1 ← new Vertex(p1)
18. h1 ← new Halfedge(v1)
19. v1.halfedge ← h1
20. v2 ← new Vertex(p2)
21. h2 ← new Halfedge(v2)
22. v2.halfedge ← h2
23. v3 ← new Vertex(p3)
24. h3 ← new Halfedge(v3)
25. v3.halfedge ← h3
26.
```

```

27. h1.next ← h2
28. h2.previous ← h1
29. h2.next ← h3
30. h3.previous ← h2
31. h3.next ← h1
32. h1.previous ← h3
33. nouvelleFace ← new Face(h1)
34. h1.face ← nouvelleFace
35. h2.face ← nouvelleFace
36. h3.face ← nouvelleFace
37.
38. ajouter(faces, nouvelleFace)
39.
40. //détection des arêtes opposées pour compléter la structure de
données
41. vertexPoint1 ← vertices.filter( element → element.point == p1
)
42. si vertexPoint1.length > 0
43.     alors
44.         halfedgeOppose ← vertexPoint1.map(element →
element.halfedge.previous)
        .filtrer(element → element.vertex.point == p2)
        .findFirst()
45.         si halfedgeOppose non null alors
46.             si halfedgeOppose.opposite est null alors
47.                 halfedgeOppose.opposite ← h1
48.                 h1.opposite ← halfedgeOppose.opposite
49.                 sinon erreur
50.             finsi
51.         finsi
52. finsi
53. //pareil pour les deux autres points
54.
55. ajouter(vertices, v1)
56. ajouter(vertices, v2)
57. ajouter(vertices, v3)
58. ftant
59.
60. points ← trierPoints(points)
61. //Trier vertex par points
62. //Trier face par vertex
63.
64. mesh ← Mesh(faces, vertices, points)
65. retourne mesh
fin

```

Pour le nouvel algorithme, nous avons notamment remplacé l'utilisation des tableaux par des SkipList. En effet, cette implémentation de liste permet un rangement ordonné des valeurs dès leurs insertions. Ceci est avantageux sur deux points principalement :

- éviter de trier la liste à l'issue de l'exécution de l'algorithme,
- gagner un temps significatif sur la recherche des valeurs dans la liste.

Et ce sans perdre de temps sur l'ajout d'une valeur dans la liste.

Cette implémentation de liste nous a également fait réduire les tailles des tableaux intermédiaires, rendant leur utilisation rapide, et a limité les appels d'attributs, qui sont à terme coûteux pour la mémoire.

Ainsi, le **nouvel** algorithme et les méthodes principales associées sont :

Algorithme remplissageDesDonnees

Fonction convertSTLToData (positions : **Tableau**[0...n]) : **Mesh**

début

```

1.  sommetsSkipList ← new VertexSkipList()
2.  faces ← [ ] // tableau de faces du modèle 3D
3.  pour i de 0 à n -1 avec un pas de 9 faire
4.      sommetsCourants ← [ ]
5.      sommetsCourants[1] ← créerSommet(new Point( positions[i],
positions[i+1],positions[i+2], sommetsSkipList))
6.      sommetsCourants[2]← créerSommet(new Point( positions[i+3],
positions[i+4], positions[i+5], sommetsSkipList))
7.      sommetsCourants[3]← créerSommet(new Point( positions[i+6],
positions[i+7], positions[i+8], sommetsSkipList))
8.      halfedges ← []
10.     pour chaque sommet de sommetsCourant faire
11.         he ← new Halfedge(sommet)
12.         halfedges.push(h)
13.     fpour
14.     pour i de 0 à halfedges.length faire
15.         he ← halfedges[i]
16.         setPrevAndNext( he, halfedges[(i+2)%3],
halfedges[(i+2)%3] )
17.     fpour
18.     pour j de 0 à sommetCourant.length faire
19.         sommet ← sommetCourant[i]
20.         halfedge ← halfedges[i]
21.         sommet.addHalfedge(halfedge)
22.     fpour
23.     face ← new Face(halfedges[0])
24.     pour chaque halfedge de halfedges faire

```

```

        halfedge.setFace(face)
25.      fpour
26.      faces.push(face)
27.      fpour
28.      halfEdgeProbleme ← sommet.getHalfEdgeProblem( )
29.      mesh ← new Mesh(faces, halfEdgeProbleme)
30.      retourne mesh
fin

```

Lexique :

- sommets : VertexSkipList, structure de données dans laquelle sont stockées les nouveaux sommets. Elle permet une recherche dichotomique des sommets
- faces : Tableau de faces, tableau dans lequel sont stockées les faces du modèles 3D
- sommetsCourant : Tableau de sommets, tableau dans lequel sont stockées les 3 sommets de la face en cours de création
- halfedges : Tableau de HalfEdge, tableau dans lequel sont stockées les HalfEdges de la face en cours de création
- face : Face, la face en cours de création

- Face <edge : HalfEdge> : Représente une Face du modèle 3D
- Vertex <point : Point, halfEdgeTab : tableau de HalfEdge> : Représente le sommet se trouvant au coordonnées *point*. Il possède également la liste des HalfEdge partant de celui-ci
- HalfEdge<vertex : Vertex, face : Face, next : HalfEdge, prev : HalfEdge, opposite : HalfEdge> : Représente une arête d'une face. Elle connaît la face à laquelle elle appartient, son sommet de départ, ainsi que l'halfedge qui la précède, qui la suit et son opposée.

```

Fonction créerSommet(point : Point, sommets : VertexSkipList) : Vertex
début
1. existingVertex ← sommets.searchVertex(point)
2. si existingVertex = null alors
3.   existingVertex = new Vertex(point)
4.   sommets.insertVertex(point, existingVertex)
5. fsi

```

```
6. retourne existingVertex
fin
```

Lexique :

- existingVertex : Vertex, le sommet à vérifier s'il existe déjà, sinon on le crée
- point : Point, le point de départ du sommet considéré
- sommets : VertexSkipList, structure de données dans laquelle sont stockées les nouveaux sommets. Elle permet une recherche dichotomique des sommets

Dans la classe Vertex :

```
Fonction addHalfEdge(he : HalfEdge)
```

```
début
```

```
1. //tailVertex permet de récupérer le sommet d'arrivée de
   l'halfedge et halfedgeTab est un tableau contenant toutes les
   halfedge partant de ce sommet
2. arrivée = he.tailVertex( )
3. //On recherche dans le sommet d'arrivée de he, une halfedge
   qui a comme point d'arrivée le sommet courant
4. opposée ← arrivée.halfedgeTab.find(halfedge =>
   halfedge.tailVertex === this)
5. si l'opposée est trouvée alors
6.     he.setOpposite(opposée)
7.     opposée.setOpposite(he)
8.     Ensuite on retire opposite du sommet d'arrivée
9.     sinon
10.    On ajoute he dans this.halfedgeTab
11. fsi
fin
```

Lexique :

- he : HalfEdge, l'halfEdge considérée
- arrivée : Vertex, le sommet d'arrivée de l'halfEdge considérée
- opposée : HalfEdge | null, l'opposée de l'halfEdge considérée, si elle existe

Dans la classe VertexSkipList :

```
Fonction searchVertex (key: Point)
début
1.   node ← this.head
2.   Tant que node différent de null faire
3.       Si non node.right ou node.right.key.compare(key) > 0
4.           alors node ← node.down
5.           sinon si node.right.key = key
6.               alors retourne node.right.value
7.               sinon node ← node.right
8.       fsi
9.   fsi
10.  ftant
11.  retourne null;
```

```
Fonction insertVertex (key : Point, value : Vertex)
Début
1.   nodes ← [ ]
2.   node ← this.head
3.   Tant que node != null faire
4.       Si non node.right ou node.right.key.compare(key) > 0) alors
5.           Nodes[0] ← node
6.           Node ← node.down
7.       Sinon
8.           Node ← node.right
9.   Fsi
11.  Ftant
12.
13.  shouldPromote ← vrai
14.  Downnode ← null
15.
16.  Tant que shouldPromote et nodes.length) faire
17.      node ← nodes[0]
18.      newNode ← new VertexNode(null, null)
19.      newNode.down ← downnode
20.      newNode.right ← node.right
21.      node.right ← newNode
22.      shouldPromote ← Math.random < 0.5
23.      downNode ← newNode
24.  ftant
```

```

25. si shouldPromote alors
26.     newHead ← new VertexNode(null, null)
27.     newHead.right ← new VertexNode(key, value)
28.     newHead.right.down ← downNode
29.     newHead.down ← this.head
30.     this.head = newHead
31. fsi
fin

```

Nous pouvons calculer la complexité des deux algorithmes afin de démontrer la différence d'efficacité de ces derniers par rapport au nombre de faces du modèle 3D importé dans l'application.

Pour cela, nous allons d'abord détailler la complexité derrière les méthodes de gestion d'une liste et d'une SkipList.

B. Calcul de complexité d'algorithmes de gestions de listes

- **Algorithme d'ajout**

Dans une SkipList :

```

fonction insertVertex (key, value, skiplist)
Début
1.     nodes ← tableau[]
2.     node ← skiplist.head
3.     Tant que node != null faire
4.         si node.right different de null ou
node.right.key.compare(key) > 0
5.             alors
6.                 nodes.unshift(node)
7.                 Node ← node.down
8.             sinon node ← node.right
9.         fsi
10.    ftant
11.    shouldPromote ← vrai
12.    downNode ← null
13.    Tant que shouldPromote = vrai et nodes.length > 0 faire
14.        node2 ← nodes.shift()

```



```

15.         newNode ← new VertextNode(key, value)
16.         newNode.down ← downNode
17.         newNode.right ← node2.right
18.         Node2.right ← newNode
19.         shouldPromote ← Math.random < 0.5
20.         downNode ← newNode
21.         ftant
22.         si souldPromote
23.             alors
24.                 newHead ← new VertexNode(null, null)
25.                 newHead.right ← new VertexNode(key, value)
26.                 newHead.right.down ← downNode
27.                 newHead.down ← skiplist.head
28.                 skiplist.head ← newHead
29.         fsi
30.         skiplist.size ← skiplist.size +1
31.         retourne skiplist
Fin

```

Complexité : $O(n)$

Dans un tableau non trié :

```

fonction push (liste, ...elements)
Début
1.     length ← liste.length
2.     pour chaque i = 0, i<elements.length faire
3.         Liste[length +i] ← elements[i]
4.     fpour
5.     retourne liste
Fin

```

Complexité : $O(n)$

- **Algorithme de recherche**

Dans une SkipList

```
Fonction searchVertex (key)
début
1.   node ← this.head
2.   Tant que node different de null faire
3.       Si non node.right ou node.right.key.compare(key) >0
4.           alors node ← node.down
5.           sinon si node.right.key = key
6.               alors retourne node.right.value
7.               sinon node ← node.right
8.           fsi
9.       fsi
10.  ftant
11.  retourne null;
12. fin
```

Complexité : $O(\log(n))$

Dans un tableau non trié :

Dans le pire des cas, l'algorithme de recherche parcourt tout le tableau.

Complexité : $O(n)$

- **Algorithme de tri**

Dans un SkipList :

Pas de méthode de tri pour une SkipList; elle est déjà triée et le reste à chaque ajout de valeur.

Dans un tableau non trié :

```
Fonction sortArr(arr)
Début
1.  pour chaque i ← 0, i < arr.length faire
2.      pour chaque j ← i, j < arr.length faire
3.          si (arr[i] > arr[j]) alors
4.              temp ← arr[i]
5.              arr[i] ← arr[j]
6.              arr[j] ← temp
7.          fsi
8.      fpour
9.  four
Fin
```

Complexité : $O(n^2)$

Ainsi, on remarque que la SkipList va “gagner” du temps de calcul sur deux éléments : la recherche de valeur et le tri, sans perdre de temps de calcul sur une autre méthode.

A présent, on peut calculer la complexité des deux algorithmes de remplissage de données.

C. Calcul de complexité d’algorithme de remplissage de données

- **Calcul de complexité du premier algorithme**

n = nombre de faces

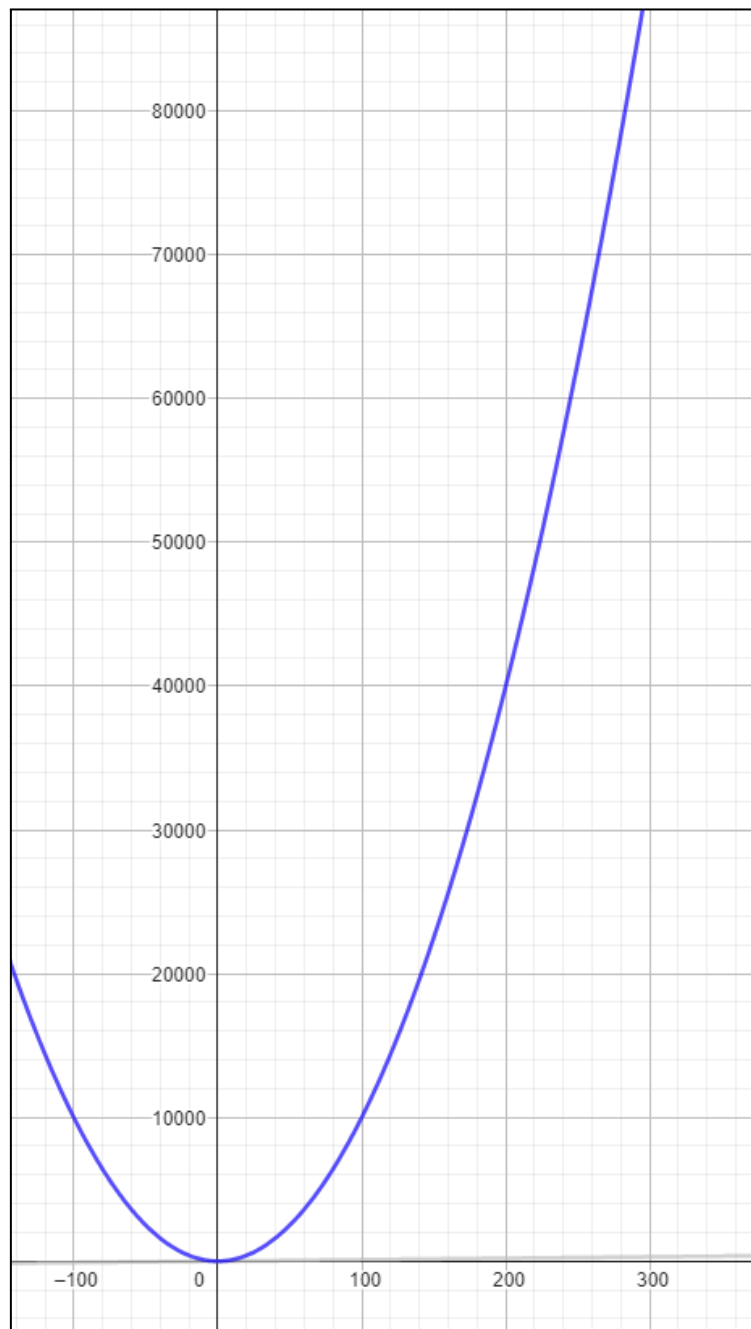
On détaille ci-dessous les lignes comportant des itérations. L’indentation des tirets représentent les boucles imbriquées

- 5 : **$9n$**
 - 11 - 15 : $3 \cdot (3n) = 9n$
 - 13 : $3 \cdot (3n) = 9n$
 - 41 : **$3n$**
 - 44 - 53 : $3 \cdot (3n + 3n) = 3 \cdot 6n = 18n$
 - 55 - 57 : $3 \cdot (3n) = 9n$
- 60 : **$3n$**

Complexité :

- De la ligne 11 à 57 les boucles ne sont pas imbriquées. Donc $O(9n + 9n + 3n + 18n + 9n) = O(51n)$. Or tous les tableaux utilisés peuvent avoir une taille infinie donc on supprime les constantes (non représentatives dans ce cas) : $O(n)$
- Les boucles de la ligne 11 à 57 sont imbriquées dans la boucle de la ligne 5. Soit $O(9n \cdot n) = O(9n^2)$. Ici encore, la constante “9” n’est pas représentative sur un très grand nombre de faces donc $O(n^2)$
- La Ligne 60 est un tri de tableau, soit une complexité $O(n^2)$
- Total : $O(2n^2) = O(n^2)$

Courbe théorique du temps d'exécution par rapport au nombre de faces :



- **Calcul de complexité du deuxième algorithme**

n = nombre de faces

Complexité de méthode préalablement calculées :

- créerSommet : $O(\log(n)) + O(n)$

De plus, les tableaux sommetsCourant et halfedges on une taille maximale de 3. Donc ses boucles sont considérées comme des constantes.

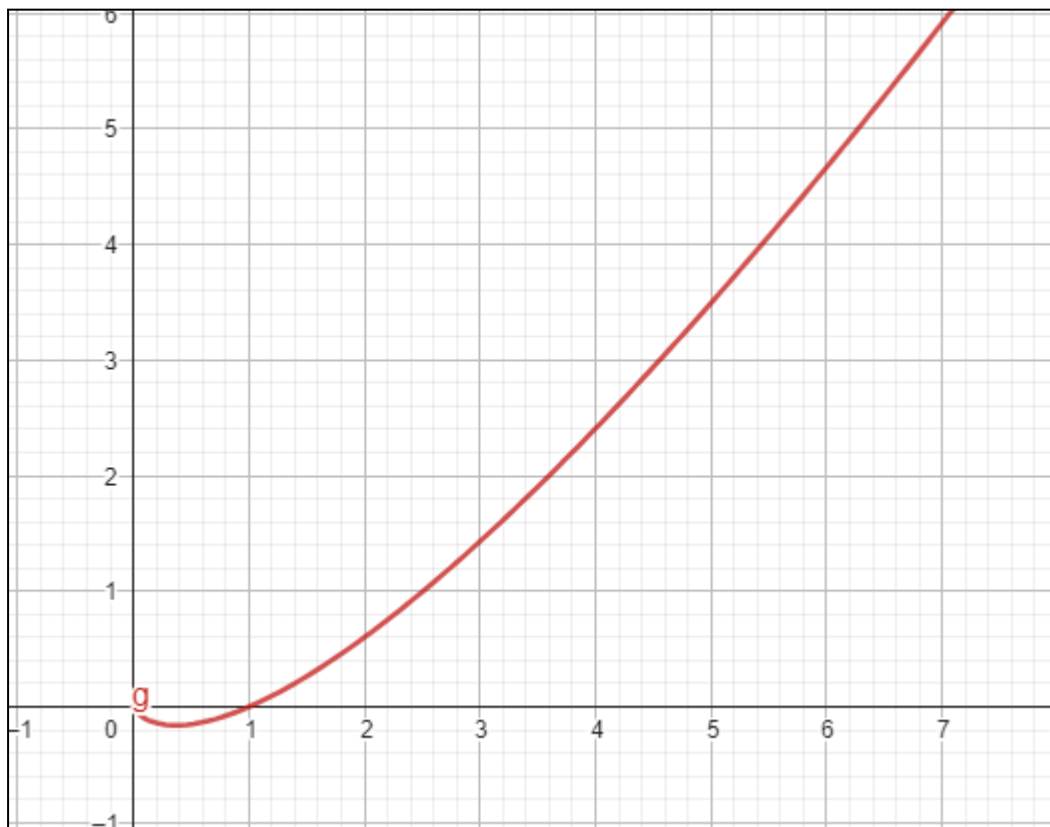
Détail des lignes avec itérations :

- 3 : $n \cdot 9$
 - 5 : $3 \cdot (O(\log(n)) + O(n))$
 - 10 : 3
 - 14 : 3
 - 18 : 3
 - 24 : 3
 - 26 : n

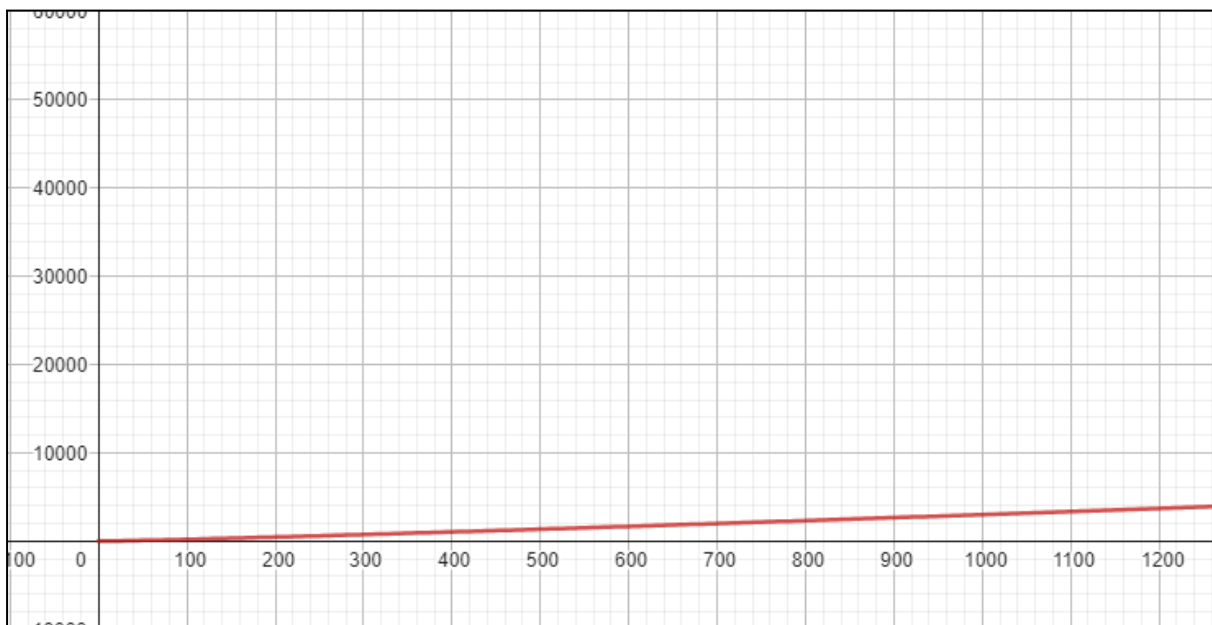
Complexité :

- De la ligne 5 à 26 il n'y a pas de boucles imbriquées.
Donc $O(3 \cdot (O(\log(n)) + O(n)) + n) = O(O(\log(n)) + n) = O(\log(n))$
- Les boucles de la ligne 5 à 24 sont imbriquées dans la boucle de la ligne 3 :
 $O(9n \cdot O(\log(n))) = O(n \cdot \log(n))$

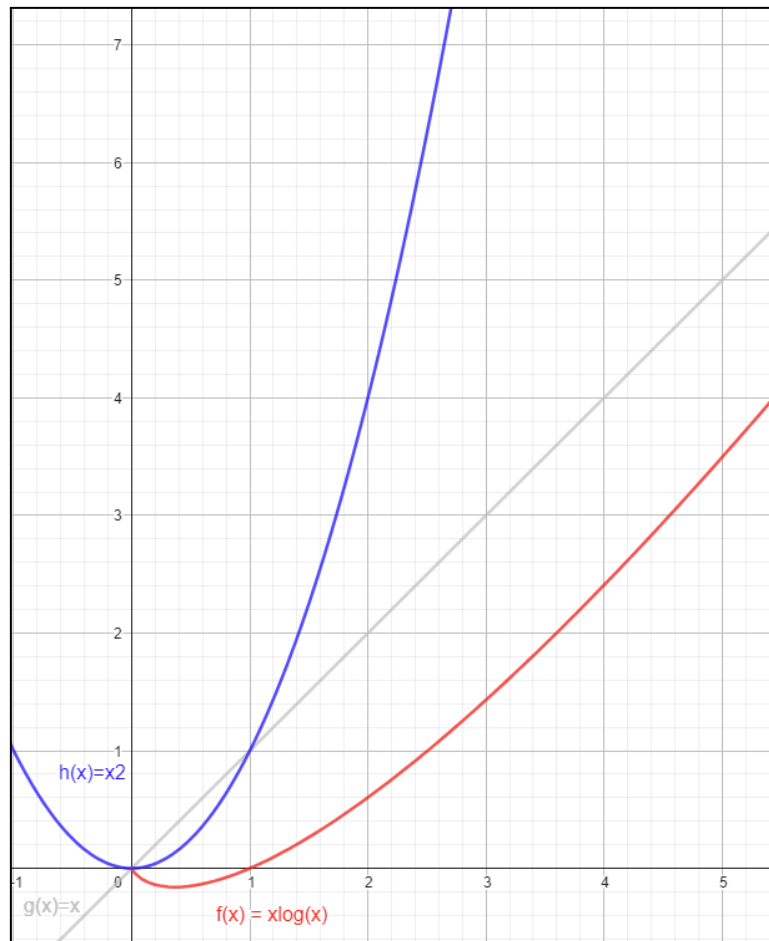
Courbe théorique du temps d'exécution par rapport au nombre de faces :



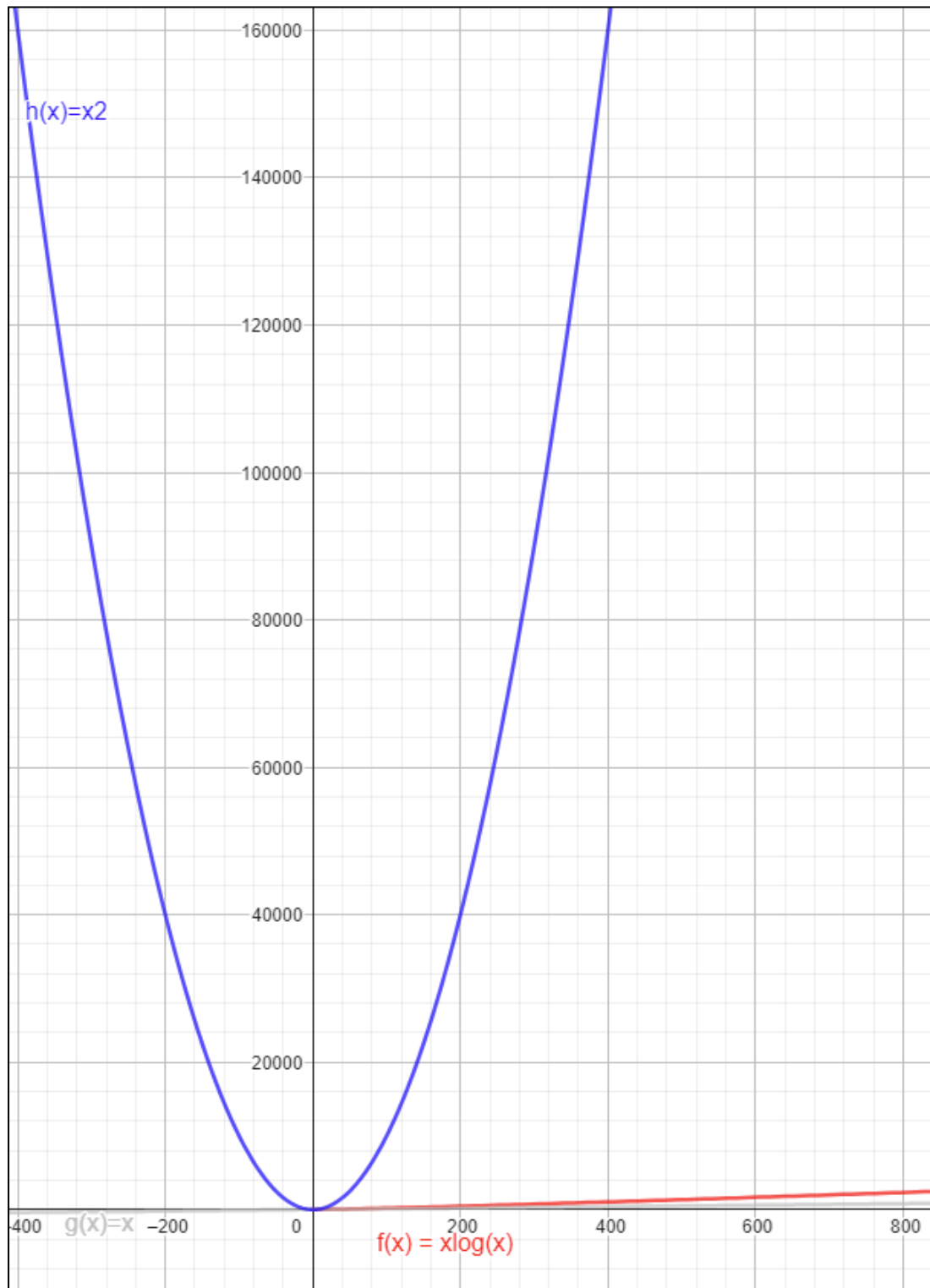
A la même échelle que la courbe bleue :



Comparaison des 2 courbes :

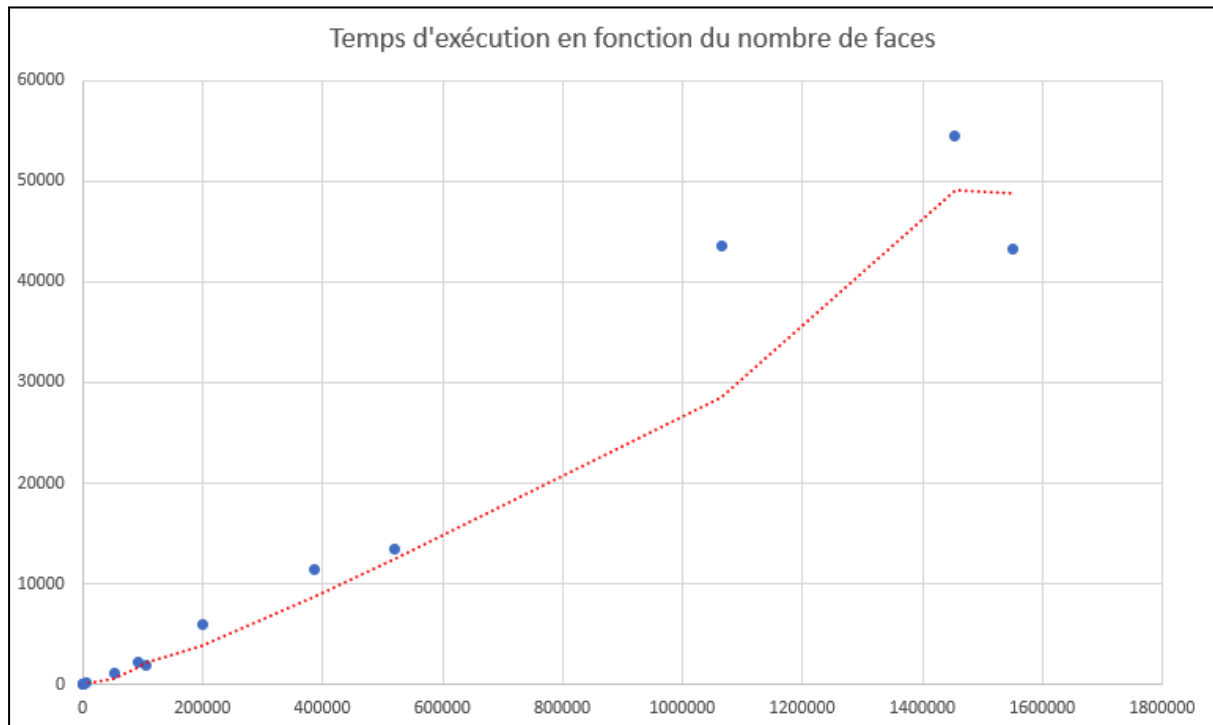


Pour un grand nombre de face, la différence théorique devient flagrante :



A présent, nous pouvons comparer ses calculs théoriques avec des mesures de temps d'exécution par rapport au nombre de face.

Pour le deuxième algorithme :



La courbe de tendance a la même forme que la courbe théorique associée, on peut donc considérer que notre calcul de complexité est correct.

Partie 2 : Réalisation du projet

I. Architecture logicielle

1. Structure HalfEdge

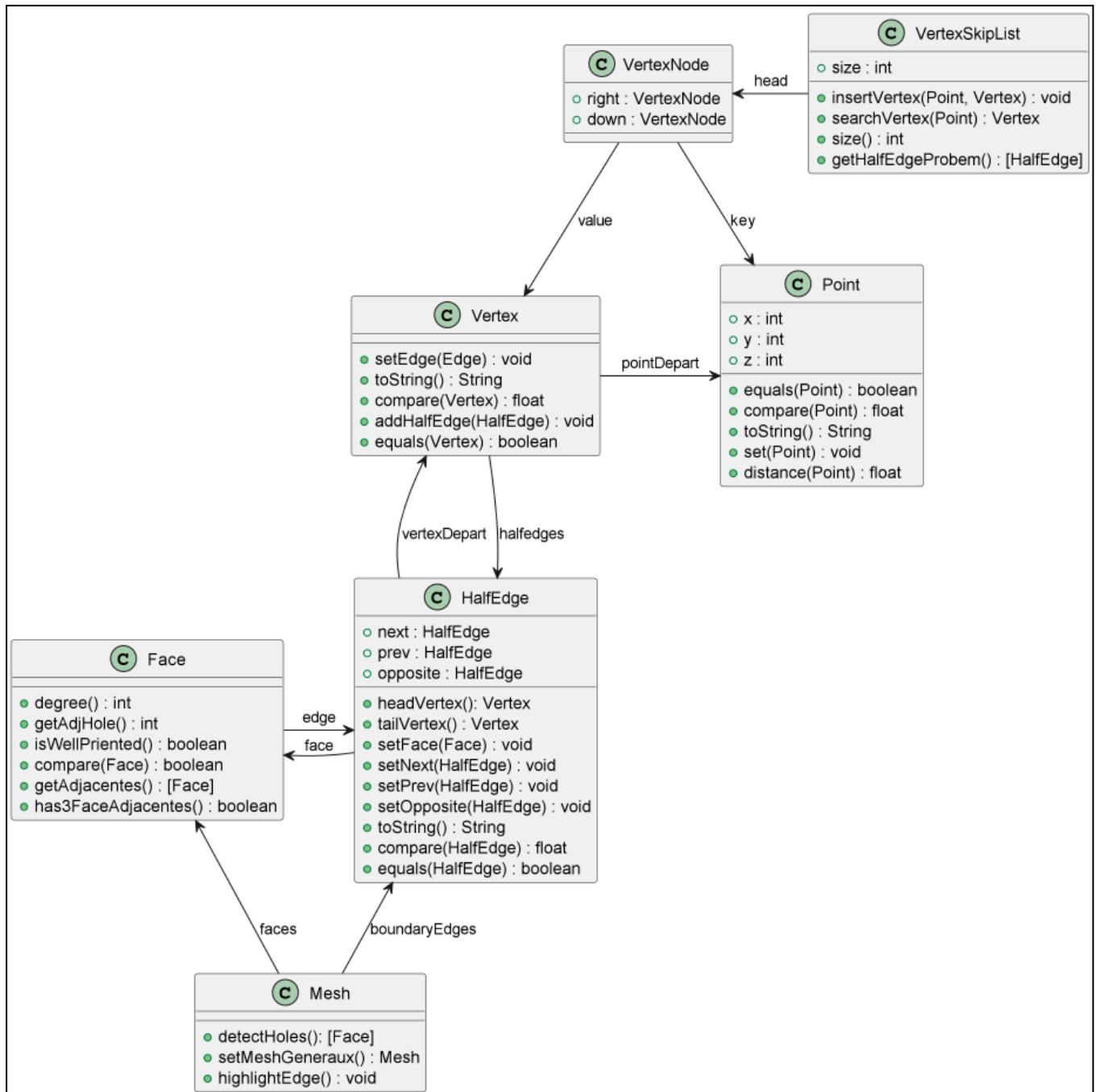


Diagramme de classe de la structure Half-Edge

La partie SkipList a été ajoutée. Aussi, un objet Mesh contient la liste des halfedge posant problèmes (*boundaryEdges*), et un sommet contient la liste des halfedge partant de lui et ne possédant pas d'halfedge opposée.

La structure de données *Half-Edge* nous permet d'avoir les informations de connectivité des faces du maillage d'un objet 3D, qui ne sont pas disponibles dans les fichiers STL. Grâce à ces informations de connectivité, nous pouvons réaliser plusieurs opérations sur ce maillage telles que la détection de trous, ou encore l'ajout d'un nouveau point sans compromettre le maillage initial.

2. SkipList

Une SkipList est une amélioration d'une liste chaînée. En effet, en plus du principe de chaînage classique, la SkipList introduit différents niveaux de chaînage supplémentaires. Chaque niveau permet "d'éviter" certains éléments du niveau précédent. C'est cet "évitement" qui rend cette structure de données plus efficace qu'un parcours de liste classique. Une "fausse" cellule permet d'accéder aux premiers éléments de la liste (non représenté sur le schéma ci-dessous). (Source : [CNRS](#))

Dans le cadre de notre structure de données HalfEdge, elle permet de rechercher et insérer rapidement un sommet.

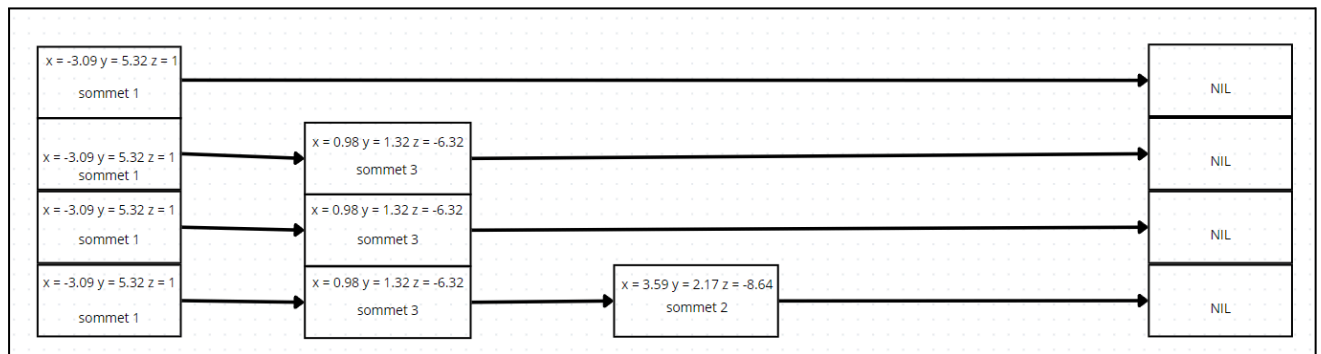
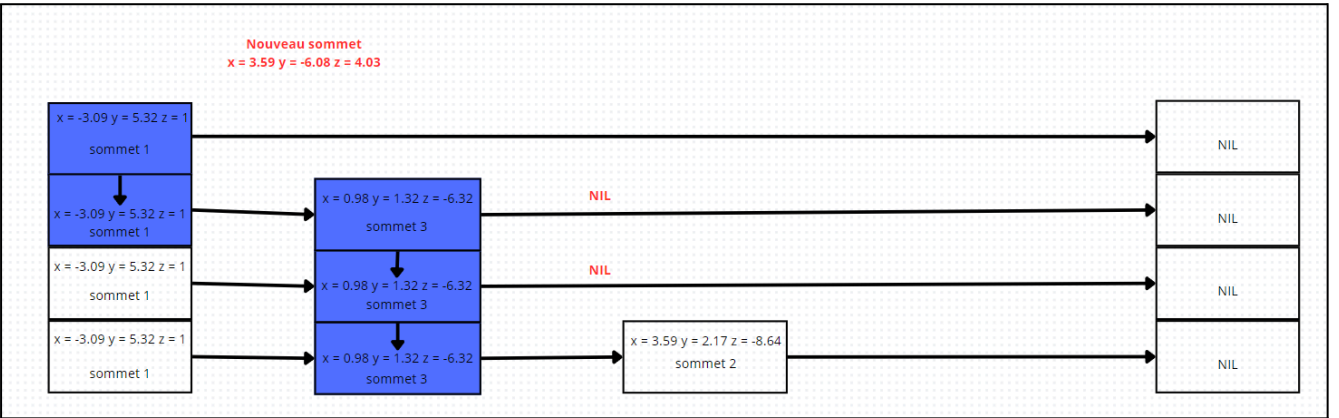
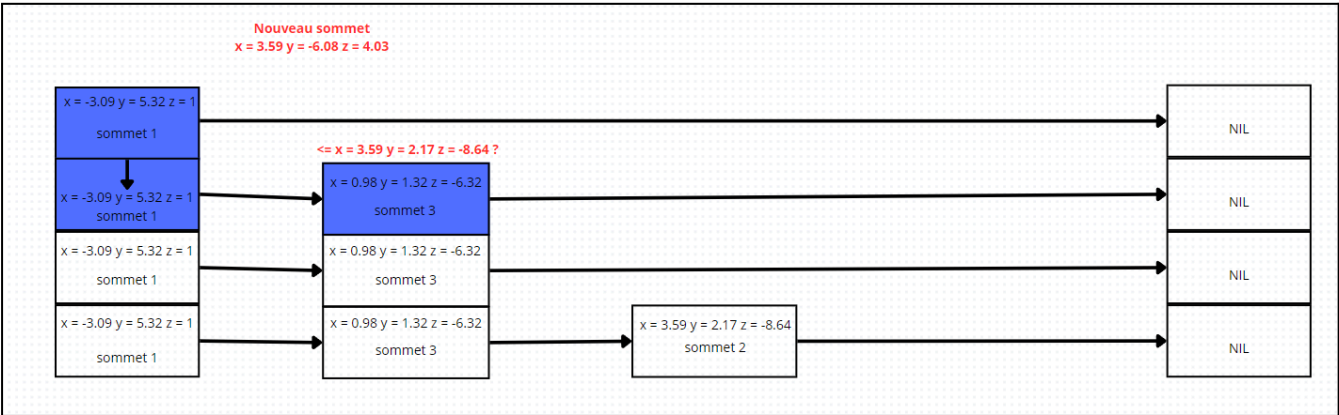
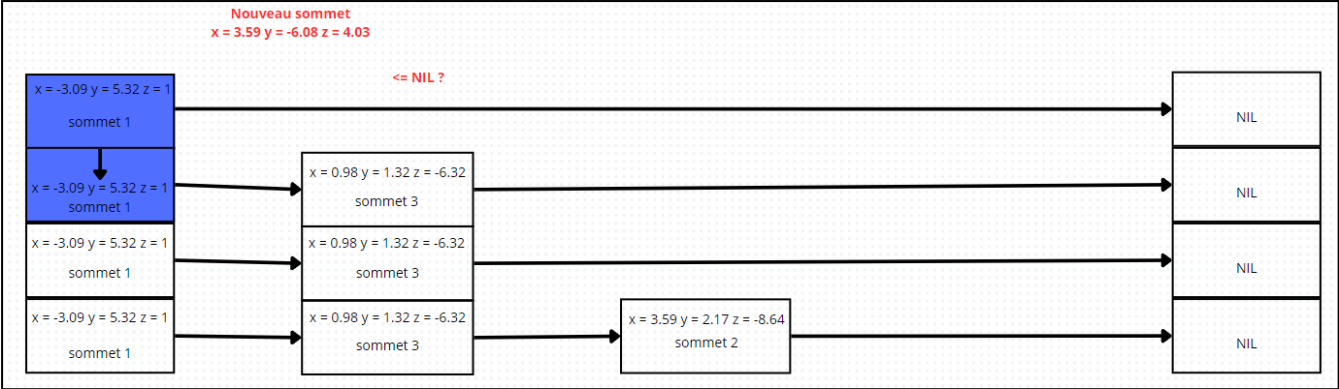
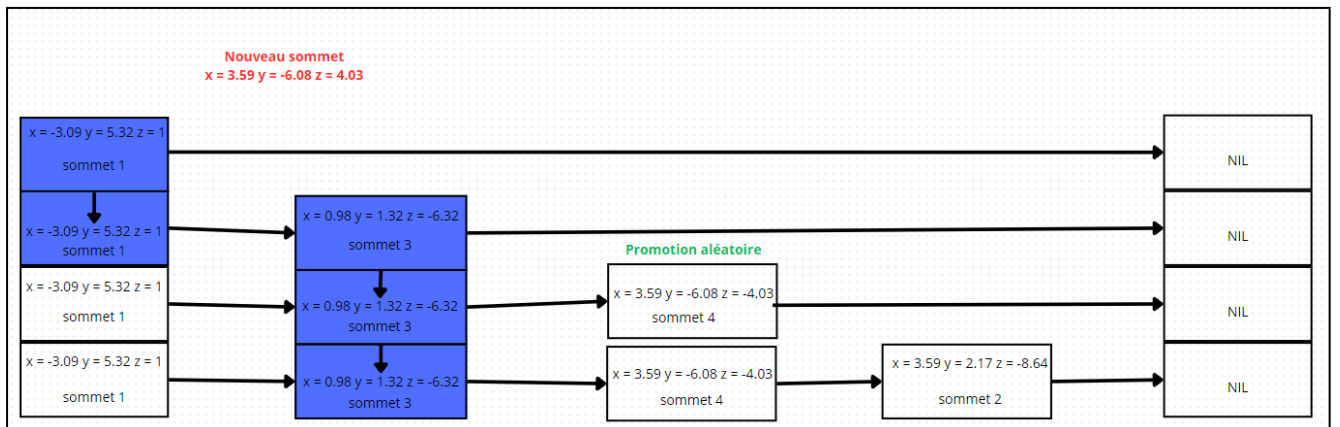
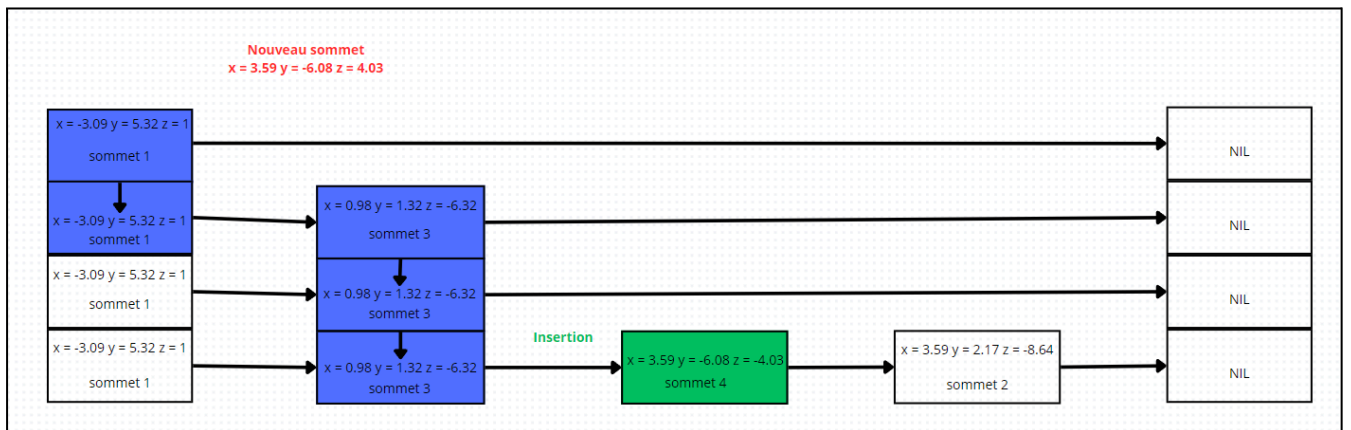
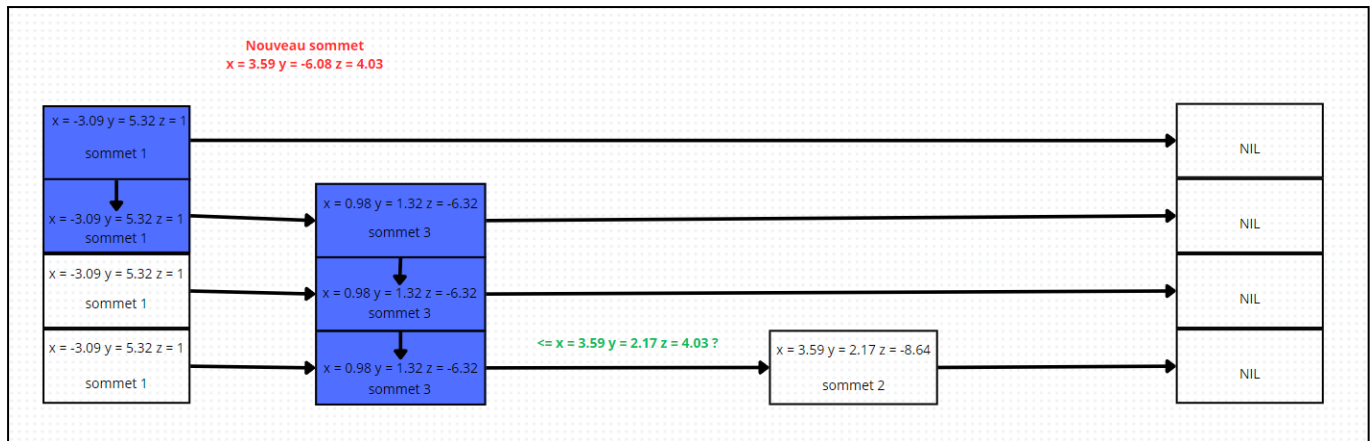


Schéma de la SkipList des sommets

Insertion dans la skipList :

Pour insérer un potentiel nouveau sommet dans notre SkipList, on vérifie d'abord qu'il n'est pas déjà dedans. Si ce n'est effectivement pas le cas, on recherche le sommet avec des coordonnées supérieures au nouveau sommet tant qu'un niveau inférieur est disponible, sinon, on l'insère à la fin. Pour cela, on compare les x, puis les y en cas d'égalité des x, puis les z en cas d'égalité des y.





La promotion d'un nouvel élément, c'est-à-dire l'ajout à un niveau supérieur, est aléatoire (pour nous cette chance est de 50%). Cela peut permettre une insertion et recherche plus rapide car certains éléments sont évités (d'où le nom de "skip").

II. Difficultés rencontrées

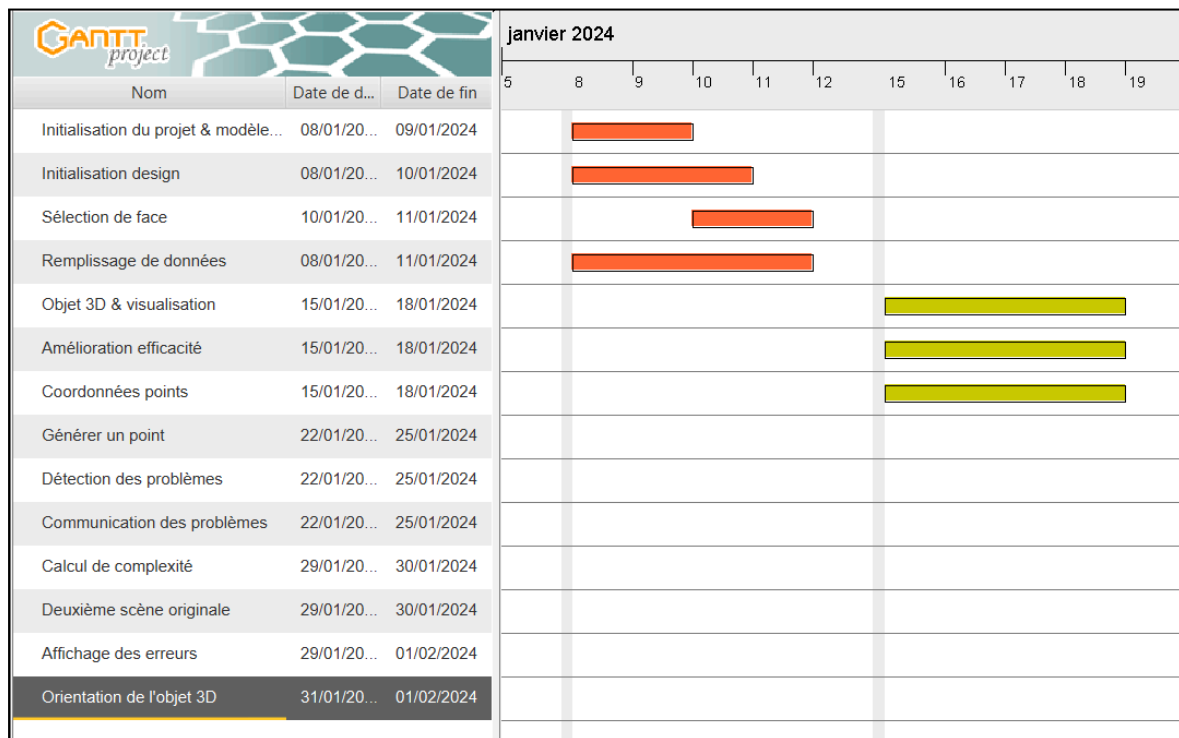
L'une des grosses difficultés que nous avons rencontrée est de pouvoir faire en sorte que le remplissage des données soit rapide et efficace. En effet, la structure de données est simple à mettre en place mais elle s'est avérée être rapidement un casse-tête lorsqu'il s'agissait de remplir ces données.

Comme nous avons pu le voir, notre premier algorithme était fonctionnel mais rapidement dépassé. Des modèles 3D de plusieurs centaines de milliers voire de millions de faces pouvaient prendre des dizaines de minutes voire même plusieurs heures dans les cas de gros modèles. Sur les conseils de M. DUPONT, nous avons implémenté la SkipList pour optimiser la recherche et la création de sommets, élément essentiel dans le remplissage de la structure. Cette nouvelle implémentation, comme nous avons pu le voir dans le calcul de complexité, nous a permis de réduire considérablement le temps nécessaire au remplissage des données, avec notamment plus que quelques minutes de chargement pour un modèle 3D de quelques millions de faces.

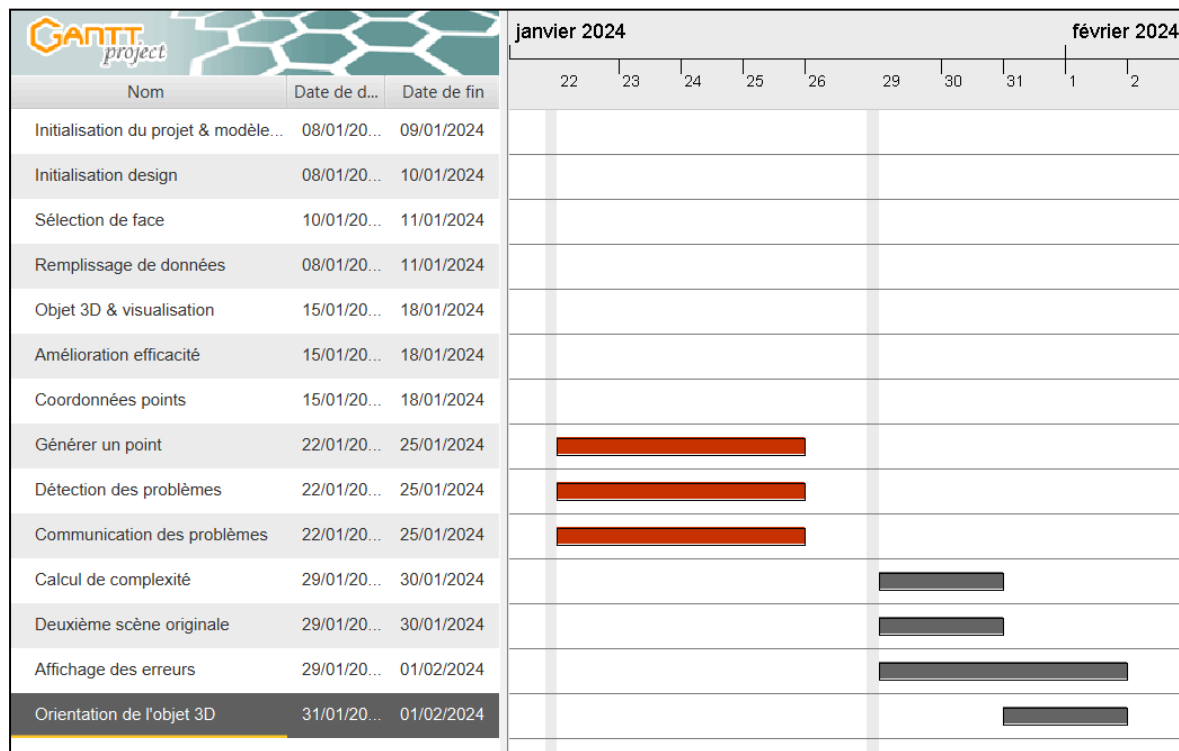
Une autre difficulté notable est la gestion et la précision des coordonnées des points. Malheureusement, ce problème n'est pas de notre ressort. En effet, la précision des points est un problème reconnu dans le monde de la géométrie 3D (il existe même des thèses qui en ont fait leur sujet). Nous devons donc faire avec et essayer de minimiser au mieux les effets indésirables que ce problème implique.

III. Déroulement du projet

1. Planning du projet



Itération 1 et 2



Itération 3 et 4

2. Répartition des tâches

	Itération 1	Itération 2	Itération 3	Itération 4
Lucie	<ul style="list-style-type: none"> - Implémentation de la structure de données (classes et remplissage) - Création menu de l'application - Bar de progression du chargement des données - Organisation visuel du menu - Refactorisation du main en module 	<ul style="list-style-type: none"> - Changement coordonnées d'un point depuis le menu de modification - Changement de coordonnées d'un point à la souris 	<ul style="list-style-type: none"> - Implémentation de la fonctionnalité permettant de générer un nouveau point ainsi que ces 3 triangles reliées aux sommets les plus proches - Ajout d'un menu contextuel pour générer un point 	<ul style="list-style-type: none"> - Calcul de complexité des algorithmes de remplissage de données dans la structure Half-Edge (avec Solène) - Correction à l'ajout d'un point : suppression de la face initiale - Rédaction des différences entre l'étude préalable et la réalisation du projet
Solène	<ul style="list-style-type: none"> - Refactorisation du remplissage des données - Création de l'écran de chargement du fichier - Implémentation méthode pour retrouver le nombre de faces adjacentes à une face donnée 	<ul style="list-style-type: none"> - Passage de la vue "fil de fer" à la vue normale de l'objet 3D via le menu de visualisation (avec Tanguy) - Affichage du relief de l'objet 3D en vue normale 	<ul style="list-style-type: none"> - Correction du problème de l'annulation des modifications de l'objet 3D lors du changement de vue - Recherche sur comment détecter les différents problèmes que l'on peut retrouver dans un objet 3D (avec Tanguy) 	<ul style="list-style-type: none"> - Calcul de complexité des algorithmes de remplissage de données dans la structure Half-Edge (avec Lucie) - Correction du diagramme d'utilisation
Xin	<ul style="list-style-type: none"> - Importation du fichier STL - Possibilité de déplacer l'objet 3D - Sélection d'une face de l'objet 3D - Affichage des sommets de l'objet 3D - Création des classes de la structure half-edge (avec Lucie) - Affichage de l'objet 3D en wireframe ("fil de fer") (avec Tanguy) 	<ul style="list-style-type: none"> - Centrer la caméra là où l'utilisateur clique sur l'objet 3D - Possibilité de configurer l'anti-aliasing via le menu de visualisation - Ajout du repère du plan qui s'adapte à la position de la caméra 	<ul style="list-style-type: none"> - Adaptation automatique de l'écran lors d'un resize - Test des algorithmes de détection des problèmes (faces mal orientées, trous et arêtes sans opposées) (avec Tanguy) - Correction problème à l'export et problème lors de l'import d'un nouveau modèle 	<ul style="list-style-type: none"> - Ajout d'une deuxième vu permettant la visualisation de l'objet initial après modification - Fonctionnalité de modification de l'orientation de l'objet 3D
Tanguy	<ul style="list-style-type: none"> - Implémentation de l'algorithme de remplissage des données (avec Lucie) - Affichage de l'objet 3D en wireframe ("fil de fer") (avec Xin) - Implémentation de l'algorithme de détection des faces mal orientées (à tester) 	<ul style="list-style-type: none"> - Amélioration de la rapidité de remplissage des données (implémentation d'une SkipList pour la recherche et la création de sommet dans la structure HalfEdge) - Exportation du modèle 3D au format STL et OBJ - Passage de la vue "fil de fer" à la vue normale de l'objet 3D via le menu de visualisation (avec Solène) 	<ul style="list-style-type: none"> - Test des algorithmes de détection des problèmes (faces mal orientées, trous et arêtes sans opposées) (avec Xin) - Recherche sur comment détecter les différents problèmes que l'on peut retrouver dans un objet 3D (avec Solène) - Communication des erreurs à l'utilisateur et affichage sur l'objet 3D 	<ul style="list-style-type: none"> - Amélioration de l'affichage des erreurs - Début de la mise en valeur des trous du maillage - Ajout et amélioration graphique des menus

IV. Présentation d'un élément original dont chaque étudiant est fier

- Lucie :

“ Durant le projet, je suis fière d'avoir pu participer à tous les “espaces” de développement de l'application. En effet, notre application a trois domaines de développements plutôt distincts : le design, la structure de données (et sa cohérence) et la gestion de l'affichage 3D. Bien que toutes complémentaires, certains d'entre nous ont pu se spécialiser dans un de ses domaines. Je suis heureuse d'avoir eu l'adaptabilité suffisante pour contribuer à trois de ces domaines et ainsi avoir une vue aussi bien générale sur l'avancée du projet que technique et pointilleuse du code.

La fonctionnalité pour laquelle j'ai été le plus en difficulté concerne la création d'un point dans la structure 3D et son intégration complète au modèle. La plupart des problèmes rencontrés durant son développement ont été en lien avec la gestion du modèle et des coordonnées de three.js. Ceci m'a poussée à étudier plus en détail son fonctionnement et la manière dont les coordonnées étaient stockées et lues afin de m'adapter et faire fonctionner mon code.

Je suis également fière d'avoir pensé et réalisé le design de l'application et des différents éléments graphiques. Bien que ce soit fait avec beaucoup de simplicité, j'ai apprécié me poser les questions quant à l'expérience utilisateur et la manière dont nous devons concevoir l'application pour que la navigation soit la plus simple possible, pour que les informations soient accessibles rapidement et ainsi laisser toute la place au sujet : la modélisation 3D. L'utilisateur doit pouvoir se concentrer uniquement sur ce point et ne pas perdre du temps ou de l'énergie dans la recherche du fonctionnement de l'application. “

- Solène :

“ Je suis personnellement plutôt fière de la fonctionnalité permettant de changer de vue (fil de fer ou matériel).

Bien que j’estime avoir passé bien trop de temps sur cette fonctionnalité pour cause de problèmes rencontrés que je n’ai pas réussi à traiter rapidement, je suis au final plutôt satisfaite du résultat et d’avoir pu au final réaliser cette fonctionnalité sans qu’elle ne soit impactée par des bugs tels que la réinitialisation des modifications lors d’un changement de vue.

Je considère cette fonctionnalité comme quelque chose dont je suis fière mais surtout dont je tire un enseignement pour les projets futurs, notamment le fait de demander de l’aide à mes camarades plus rapidement dans le cas où je serais totalement bloquée, ce que je n’ai pas réalisé lors de l’implémentation de cette fonctionnalité, mais qui aurait très certainement pu accélérer la réalisation de ce changement de vue. “

- Xin :

“ Dans le projet, j’ai surtout travaillé sur les interactions entre les utilisateurs et les objets 3D dans Three.js, et sur la correction des bugs du programme.

Parmi les fonctionnalités que j’ai implémentées, il n’y avait rien de très difficile dans ce que j’ai fait, mais je suis fier des problèmes compliqués que j’ai réussi à résoudre.

Pour mettre en œuvre une fonctionnalité ou corriger une erreur dans le programme, je devais souvent consulter la documentation de Three.js et rechercher dans les forums si d’autres avaient implémenté des fonctionnalités similaires.

En repensant aux dernières itérations, j’ai résolu un problème après l’autre, et ces succès sont suffisants pour me satisfaire. ”

- Tanguy :

“ Personnellement, je suis assez fier de mon implémentation de la structure de données HalfEdge. Cette structure de données est un point clé de notre application et n’a pas été simple à mettre en place.

Cette structure de données est simplement une représentation du modèle 3D sous forme d’objet représentant les faces, sommets et arêtes du modèle 3D. Elle nous permet de vérifier la bonne cohérence du maillage du modèle et d’en détecter les incohérences. Cette application étant destinée à aider les utilisateurs pour l’impression 3D, la détection de ces incohérences est donc essentielle.

Une première version de notre algorithme prenait quelques secondes à charger les données de petits objets 3D (moins de 1 000 faces) mais pouvait prendre des dizaines de minutes pour un objet de 100 000 faces et pouvait monter à plusieurs heures pour un objet à 1 ou plusieurs millions de faces.

J’ai pu réduire ce temps à quelques minutes grâce à l’implémentation d’une SkipList pour la création et la recherche de sommet. Cette optimisation a permis à notre algorithme d’être beaucoup plus rapide et efficace. Elle permet également de gérer la non-duplication des sommets, d’une part pour minimiser au mieux la mémoire utilisée par l’application mais aussi pour que lors de la modification de la position d’un sommet, il n’y ait qu’un objet à modifier et non plusieurs dans notre structure de données HalfEdge. ”

V. Objectifs pour la fin du projet

L'objectif de cette fin de projet va se concentrer sur la finalisation de la détection des erreurs et la réparation du maillage. Ces points sont les deux derniers à implémenter pour contenter toutes les demandes du projet. De plus, il est prévu de donner l'emplacement des triangles à problèmes dans le fichier STL, ce qui permettra à l'utilisateur de vérifier s'il y a bien un problème et le cas échéant, le régler. Cet ajout permettrait de compenser notre problème de précision des coordonnées des points.

D'autres objectifs annexes viennent s'ajouter dans le but d'améliorer l'expérience utilisateur tels que la sélection de plusieurs face du modèle 3D ou encore la séparation du chargement du modèle et des données (pour un rendu plus rapide). Aussi, un objectif facultatif mais qui serait très intéressant comme expérience pour nous, serait le portage de notre application en un format de réalité virtuelle. Cela ne demanderait qu'à changer les contrôles d'interactions au sein de l'application ainsi qu'étudier three.js pour l'adapter au mieux à ce nouvel environnement.

En résumé, l'objectif pour la fin du projet est d'avoir une application permettant la visualisation et la modification de modèles 3D, tel que demandé par le sujet. Cette application devra être totalement fonctionnelle, avec des délais de chargement convenables ainsi que le moins de bugs ou autres problèmes possibles. Enfin, si le temps nous le permet, nous aimerions faire un portage de notre application dans un environnement de réalité virtuel.

VI. Planning pour les deux itérations restantes

Comme vu précédemment, il y a encore des choses à faire et qui vont être répartis comme suit dans les deux dernière itérations :

Itération 5 (~20h):

- Afficher les trous du maillage peu importe le nombre de triangle qui le forment (pour l'instant, nous n'arrivons à gérer que des trous d'un seul triangle).
- Afficher l'emplacement des problèmes dans le fichier STL dans notre menu "Problèmes"
- Afficher les faces qui se croisent.
- Commencer l'implémentation de réparation automatique du maillage

Itération 6 (~20h):

- Sélection de plusieurs faces du modèle 3D (indépendant de la structure de données pour le moment)
- Terminer la réparation automatique du maillage
- Si il reste du temps:
 - Aborder la fonctionnalité de visualisation d'un objet 3D en VR
 - Revoir séparation du chargement du visuel et de la structure de données
- Préparations à la soutenance finale du projet

Conclusion

Ce projet, bien qu'il ne soit pas encore totalement abouti, peut déjà effectuer une majorité des fonctionnalités demandés. Au fil des itérations, des changements ont eu lieu par rapport à l'étude préalable dans un souci de facilité et/ou d'efficacité. Les itérations 5 et 6 vont nous permettre de finaliser au mieux notre projet.

Nous avons pu rencontrer quelques problèmes que nous avons réussi à résoudre, au mieux les minimiser, mais d'autres sont encore en attente d'une solution, ce qui sera également un point important des 2 dernières itérations à venir.

En termes d'humain, ce projet est le fruit d'une collaboration étroite entre tous les membres du groupe, chacun à son échelle. Chacun a effectué ses tâches assignées, avec l'aide d'autres si besoin. C'est aussi le fruit d'une réflexion collective lorsque certains problèmes empêchant l'avancée du projet ne pouvait pas être résolu par une seule personne.

Enfin, ce projet nous permet de découvrir la technologie **three.js**, permettant de manipuler des objets 3D, de faire un premier pas dans le domaine du calcul de complexité d'algorithmes et de découvrir de nouvelles structures de données que sont HalfEdge et SkipList.