Géométrie Numérique

Segmentation de maillages

Méline Bour-Lang, Tiphaine Richard, Morgane Ritter & Nathan Roth

A propos de l'algorithme

Déroulement

Pour segmenter des maillages de plantes, nous avons opté pour une méthode se basant sur la squelettisation d'un modèle 3D.
Nous nous sommes plus particulièrement appuyés sur *Decomposing Polygon Meshes for Interactive Applications* [1], qui propose de découper la segmentation en deux étapes.

La première consiste à squelettiser le modèle. Les os résultants sont ensuite utilisés comme base pour parcourir le maillage : un plan, dont la normale suit l'os, avance le long de celui-ci. L'intersection de ce plan avec le mesh permet d'en déduire des caractéristiques : si la topologie change, on définit qu'on a atteint un point critique, et un nouveau composant est généré.
A la fin de ces deux algorithmes, le maillage est donc découpé en plusieurs composants.

La squelettisation en détail

La squelettisation étant une opération destructive, avant de la lancer, on veille à sauvegarder les informations nécessaires (arêtes, sommets...) dans l'objet représentant le maillage.
Pour squelettiser le maillage, on procède par élimination d'arêtes : afin de limiter l'erreur produite, on choisit toujours de supprimer l'arête la plus courte. Sa suppression va aplatir les faces qui lui étaient incidentes ; ces faces seront remplacées par des arêtes, et mémorisées par elles dans une liste ATL (*Associated Triangle List*).

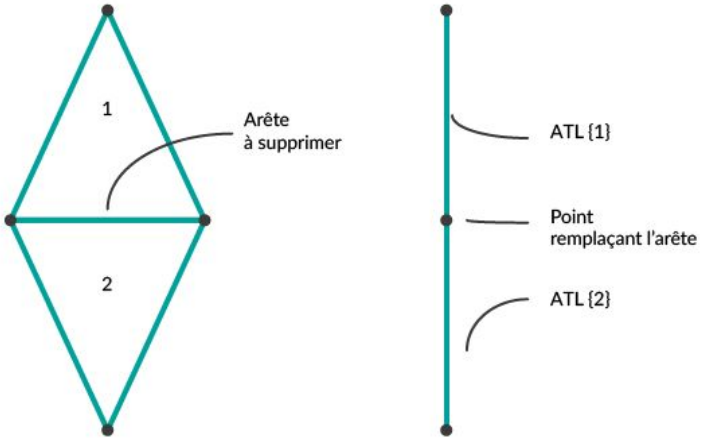

Figure 1 : construction des listes ATL à la destruction d'une arête

Lorsqu'une arête n'est plus incidente à aucun triangle, on déclare qu'elle fait partie des os du squelette et qu'elle, et ses sommets, seront protégés de la suppression. Il est nécessaire de s'assurer que le squelette ne soit pas disjoint : [1] propose de créer des *virtual edges*. Ainsi, lorsque l'un des sommets de l'arête étudiée, dans notre exemple l'arête (1,2), est protégé, nous le relions par une virtual edge au sommet créé par la suppression de cette arête.

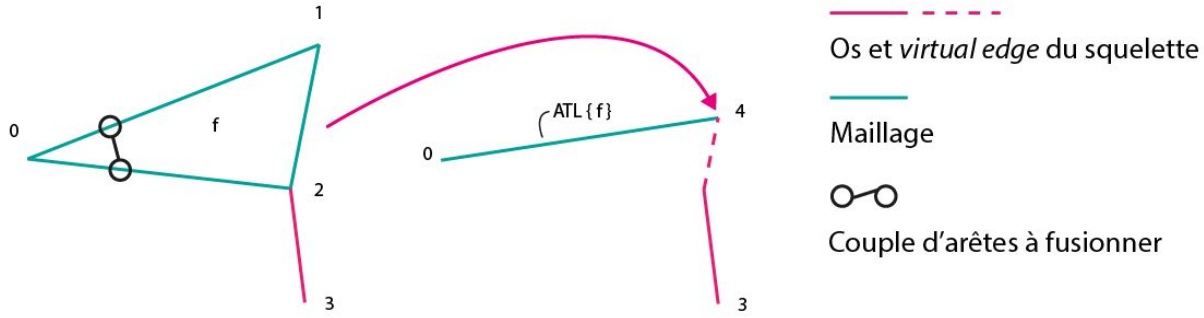

Figure 2 : création d'une virtual edge

Le résultat de cette étape est un ensemble d'arêtes liées entre elles, contenant des listes ATL.

Segmentation : détail

La segmentation utilise la squelettisation pour segmenter le maillage : on travaille donc sur les deux objets en parallèle.

Les listes ATL vont permettre de trier les os du squelette, afin de savoir dans quel ordre les traiter. Ainsi, chaque liste ATL est constituée de faces, dont l'aire totale est le coût associé à l'os. On traitera en premier les arêtes associées aux aires les plus petites : commencer par celles-ci permet de s'assurer que les plus petits composants ne soient pas absorbés par les plus gros.
On étudie ensuite les os les uns après les autres, grâce à un plan qui va les parcourir en suivant leur normale.

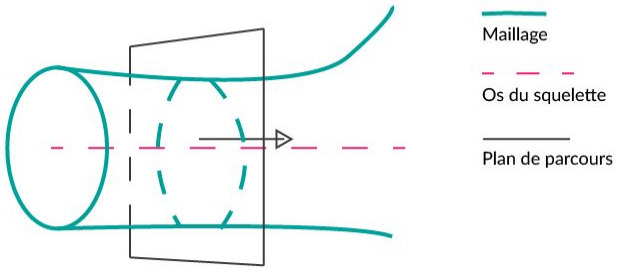

Figure 3 : plan de parcours

Ce plan intersecte le maillage : c'est cette intersection que nous allons étudier pour savoir quand commencer une nouvelle section.

Cette intersection (*cross-section*) peut former un ou plusieurs polygones, dont on va étudier le périmètre et ses dérivées (première, seconde et troisième). On compte également le nombre de polygones simples qui composent la *cross-section*. En fonction des changements observés, on est en mesure de définir si le plan est à la frontière entre deux composants.

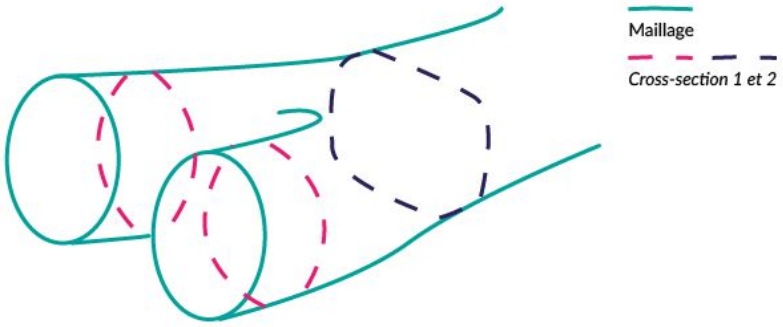

Figure 4 : deux intersections possibles du plan de parcours avec le maillage

Cet algorithme de segmentation n'as pas pu être implémenté par manque de temps. Nous avons choisi d'implémenter une version simplifiée n'utilisant aucune intersection. En effet, à l'issue de la squelettisation, les os et les virtual edges contiennent les listes ATL des faces qui se sont effondrées sur ces arêtes. Il est donc justifié de considérer que ces faces ont contribué à la création de l'os/de la virtual edge, et font donc partie de l'élément auquel appartient l'os/la virtual edge.

Nous avons choisi de définir un élément comme étant l'ensemble des arêtes du squelette comprises entre deux noeuds de ce dernier.

Cette segmentation découpe donc l'objet en fonction de la topologie de son squelette, non pas en fonction des "features" de l'objet, comme des feuilles ou des branches. Il est possible de pallier à cette limitation en regroupant les sommets des faces de plusieurs éléments en fonction d'une analyse ACP, qui nous permettrai de définir si un ensemble est aplatie (une feuille) ou longiligne (une branche).

Points particuliers

Problème du prisme

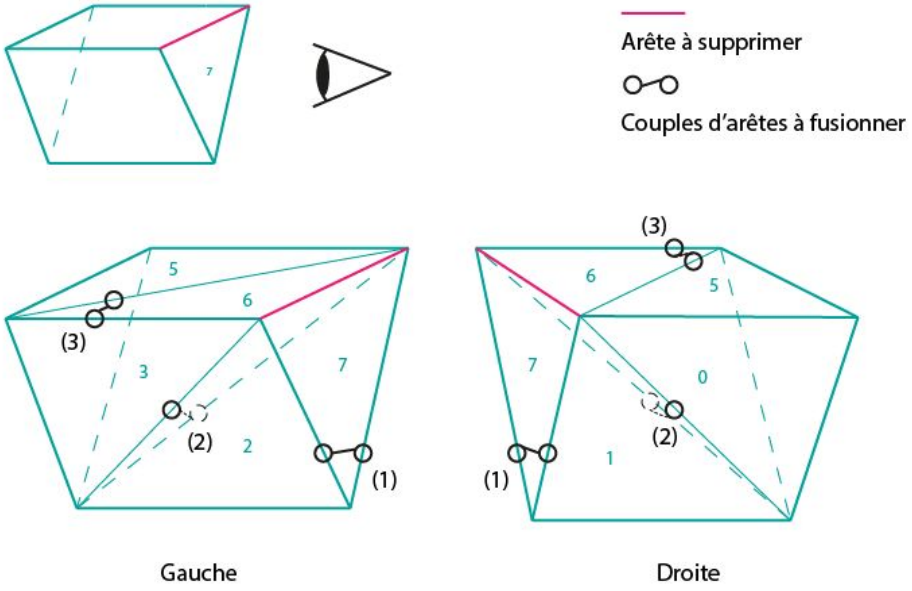

Figure 5 : arêtes à fusionner dans le cas d'un prisme

Lorsque l'on supprime une arête d'un prisme en suivant la configuration ci-dessus, on constate que trois couples d'arêtes doivent être supprimés. En effet, ces arêtes sont devenues redondantes.
L'algorithme de squelettisation que nous avions implémenté gérait correctement les couples (1) et (3). En effet, ces deux cas partageait une face en commun, contrairement au couple (2). Pour gérer celui-ci nous avons dû nous résoudre à implémenter un traitement spécial. Nous déterminons si les edges du couple (2) possèdent dans leur liste de faces les deux faces non-supprimées d'un autre couple, ici (7). On a les faces 2 et 3 ainsi que 1 et 0 connectées au couple (2), et les faces 2,7 et 1 dans le couple (1). 2 et 1 apparaissent 2 fois, nous avons donc un cas spéciale et sommes en mesure de prédire la superposition des faces 2 et 1. L'une des deux est alors supprimée puis ajoutée dans la liste ATL de l'arête survivante du couple (2). Cette dernière sera alors connectée à trois faces : 2 (ou 1), 3 et 0.

Problème du tétraèdre

Lorsque nous cherchons à squelettiser un tétraèdre, et que l'on souhaite supprimer une de ses arêtes, cela supprime deux faces. Les deux faces restantes sont confondues : lorsque l'on continue l'algorithme, la squelettisation se comporte comme prévue sur une des deux faces, et l'algorithme s'arrête. La face dédoublée subsiste, ce qui pose problème.

Pour le résoudre, nous comparons les listes des faces connectées des arêtes survivantes. Si deux arêtes possèdent deux faces identiques, alors ces deux faces sont superposées. L'une d'elle est alors supprimée puis insérée dans la liste ATL d'une des arête.

Intersections

Nous avons rapidement réalisé que le calcul d'intersections pouvait poser de grands problèmes de performances. Lorsque nous voulons réaliser la *cross-section*, nous cherchons l'intersection entre le maillage et le plan de parcours ; or, pour des maillages réalistes, le nombre d'arêtes à tester est très important.
Nous avons donc proposé une solution : l'os sur lequel le plan de parcours navigue contient une liste de faces (ATL). Ces faces devraient vraisemblablement être les seules à potentiellement intersecter le plan de parcours, ce qui devrait nous permettre de tester beaucoup moins d'arêtes (dans le cas d'un squelette composé d'os d'une longueur relativement courte).

Utiliser les listes ATL est également pertinent lorsque l'on forme un composant. En effet, le plan détecte les emplacements où le maillage change suffisamment pour en faire une nouvelle section. A ce moment, il est encore nécessaire de récupérer toutes les faces précédemment parcourues. Pour cela, on cherche à récupérer toutes les faces se situant *en dessous* de notre plan de parcours. On ne teste donc que les faces appartenant aux ATL, ce qui permet également de ne pas récupérer des faces appartenant à un autre os.

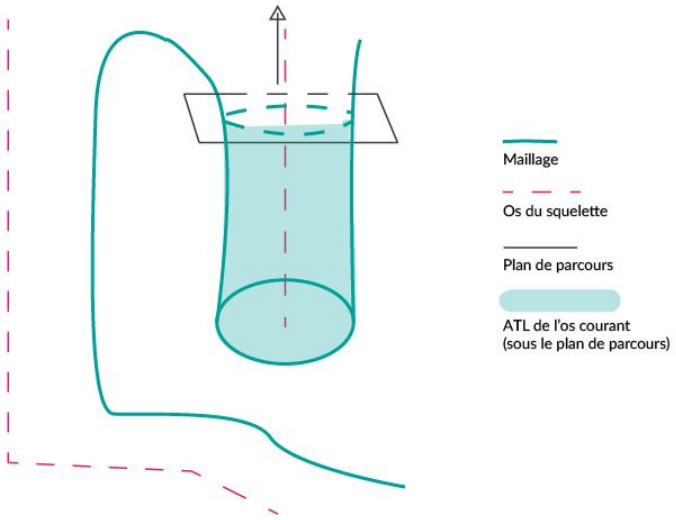

Figure 6 : Cas où récupérer la liste ATL permet de former un composant cohérent

Dérivées première, seconde et troisième

Comme expliqué ci-dessus, les dérivées font partie des caractéristiques utilisées pour déterminer si le plan se situe à la frontière entre deux composants significatifs du maillage.

La dérivée première calcule le taux de variation du périmètre entre deux emplacements du plan de parcours. On calcule ensuite la dérivée seconde, qui pour en calculer la variation, nécessite trois valeurs. Enfin, la dérivée troisième demande cinq valeurs.

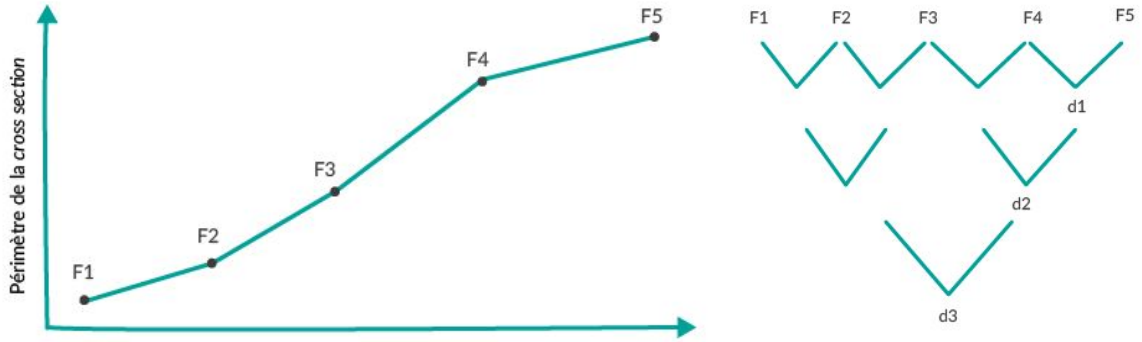

Figure 7 : dérivées première, seconde et troisième

Choix d'implémentation

Structure de données pour le maillage

Au départ, nous avions repris la structure half-edge implémentée lors des TPs, mais cela rendait chaque étape plus compliquée, puisqu'il fallait gérer chaque arête dans les deux sens. Nous avons donc décidé d'implémenter notre propre structure.

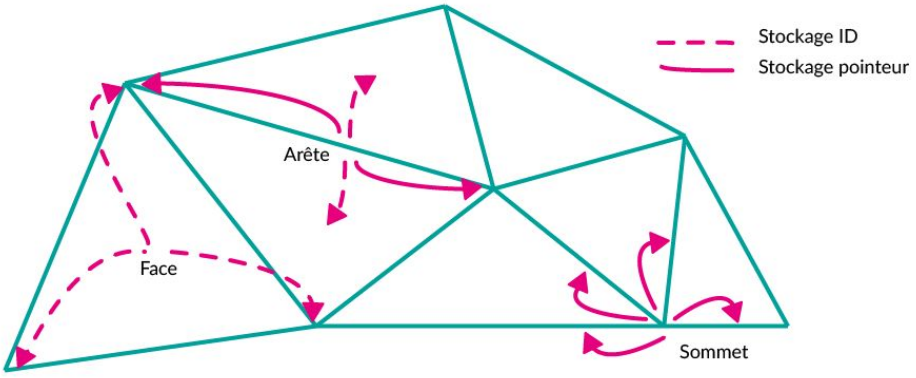

Figure 8 : structure implémentée

Stockage réalisé par une face : IDs des sommets incidents (positions)
Par une arête : sommets (pointeur vers l'objet) et IDs des faces incidentes
Par un sommet : arêtes (pointeur vers l'objet) incidentes

Organisation du code

Pour ce qui est de l'organisation du code, nous avons donc fonctionné avec ces classes, ainsi qu'une classe dédiée au maillage, et une autre au plan de parcours. La squelettisation étant une part conséquente du projet, nous l'avons déplacée dans un fichier à part ; les fonctions qui ne sont propres à aucune classe sont dans un document *utils*.

Un dossier *MeshTest* comporte les maillages qui nous ont permis de tester notre algorithme avant de passer aux structures plus complexes.
Nous avons également fait en sorte que l'export soit disponible au format OBJ, plus facile à visualiser.

Dans l'algorithme que nous avons choisi, il était proposé une étape de suppression des cycles dans le squelette. Nous avons décidé d'ignorer cette suppression, étant donné que nous ne devrions normalement pas rencontrer de cycle avec les plantes.

Le document qui nous sert de référence propose également de corriger l'orientation du plan ; nous avons ignoré cette étape pour simplifier nos calculs.

Importation et exportation de fichiers

Nous avons choisi de permettre, en plus de l'importation des fichiers OFF, nécessaires pour le traitement des plantes, l'importation des fichiers de type OBJ. En effet, afin de tester efficacement le code nous avons dû produire des objet de test. Ces objets étaient modélisés sur Blender, qui permet l'exportation sous de nombreux formats, dont OBJ, mais pas OFF. OBJ ayant une structure très similaire à OFF, nous avons choisi ce format pour nos tests.

Les fichiers de type OFF et OBJ ne décrivent le mesh que sous la forme d'une liste de face. Or nous avons besoin de la liste des arêtes du maillage. Pour la générer avec notre structure, nous extrayons les 3 arêtes de chaque face, puis cherchons dans une map si elles existent déjà. On ajoute l'arête à la map si elle n'existe pas. L'utilisation d'une map (std::map) est très importante ici. Une map accède à ses données via une clé (ici, une paire d'entier, les ID des sommets de l'arête), et utilise cette clé pour les stocker de manière ordonnée dans un arbre. Cette structure permet une recherche très rapide des éléments, et nous permet d'importer les fichiers de manière quasiment instantanée. La première implémentation utilisait contrairement à une structure en liste, obligeant à parcourir entièrement la liste à chaque requête d'insertion dans le pire des cas.

Nous avons également créé plusieurs fonctions d'exportation de nos résultats sous le format OBJ afin de pouvoir les visualiser sous blender. Il nous est ainsi possible de visualiser notre segmentation finale ainsi que notre squelette.

Evaluation de l'algorithme

Nous avons tout d'abord travaillé en 2D : en effet, notre document de référence proposait un exemple de squelettisation. Nous avons reproduit le maillage, très simple, d'origine, et l'avons donné en entrée à notre algorithme. Cela nous a permis de corriger la squelettisation, et de nous assurer que l'évolution du maillage, ainsi que la sortie proposée était corrects.
Nous avons réalisé quelques exemples 2D supplémentaires, puis nous sommes passés à un modèle simplifié en 3D dont on peut trouver le déroulement en annexes.

Nous avons segmenté un modèle à la main pour nous servir de référence quant au résultat final que nous souhaitions obtenir :

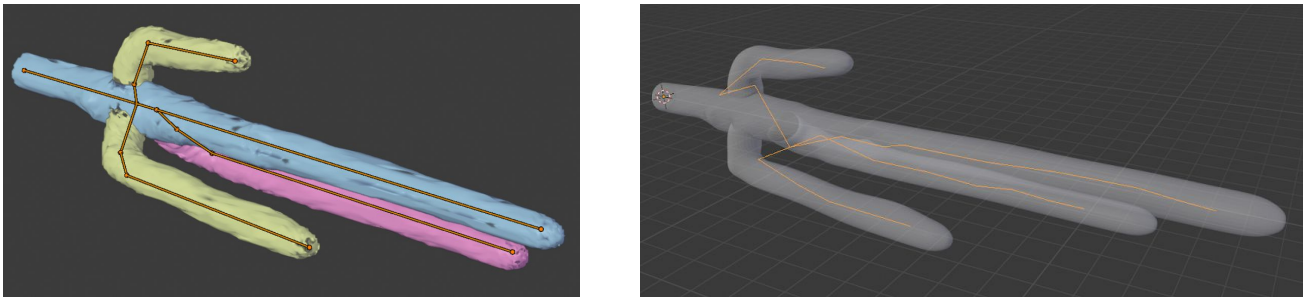

Figure 9 : Modèle squelettisé et segmenté manuellement à gauche, le même modèle squelettisé à l'aide de notre algorithme

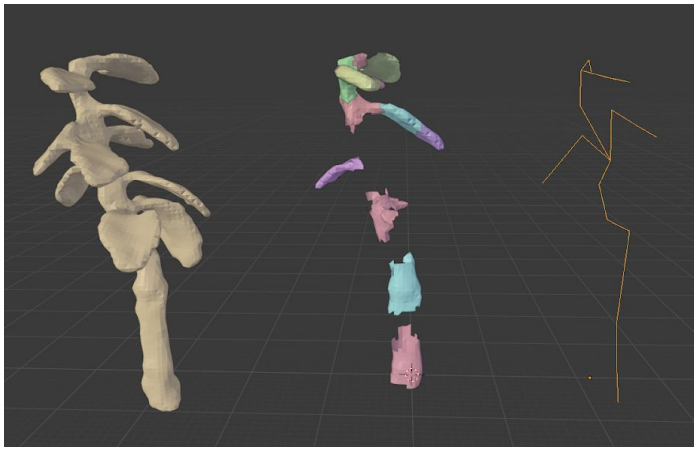

Figure 10 : Modèle segmenté automatiquement (de gauche à droite : modèle original, modèle segmenté, squelette du modèle)

Maillage	Nombre de faces initial	Nombre de faces après la squelettisation	Temps d'exécution	Segmentation	Pourcentage de faces conservé
Plante1	16540	5519	0m37	Marche	33,36759371
Plante2	25200	4438	1m9	Ne marche pas	17,61111111
Plante3	18336	10349	0m50	Ne marche pas	56,44088133
Plante4	20524	16613	0m58	Ne marche pas	80,94426038
Plante5	32200	16216	2m42	Non testé	50,36024845
Plante6	13886	8063	0m25	Non testé	58,06567766
Plante7	138418	66111	76m6	Non testé	47,76185178
Plante8	19470	10922	0m52	Non testé	56,09655881
Plante9	39048	16418	4m13	Non testé	42,04568736
Plante10	95016	35740	32m57	Non testé	37,61471752
				Moyenne	49,18820784

Figure 11 : Tableau récapitulatif des performances des algorithmes

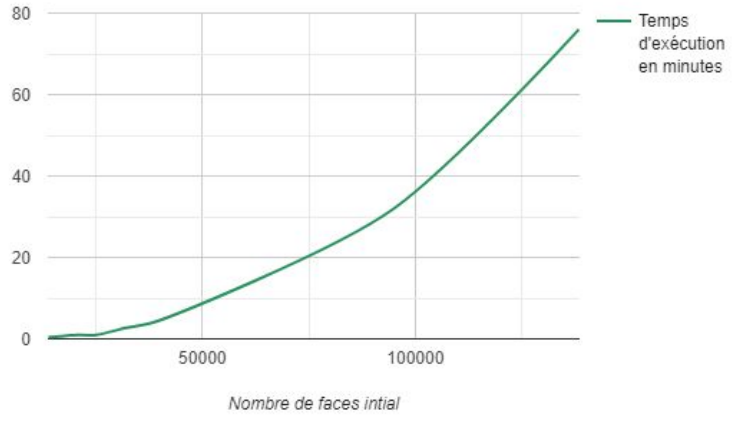

Figure 12 : Temps d'exécution en fonction du nombre de faces initials

Analyse critique

Nous avons choisi cet algorithme car il nous a semblé particulièrement adapté aux formes des plantes : la squelettisation devrait très bien restituer la structure de ces maillages.
Nous avons également trouvé son fonctionnement intuitif, puisqu'il est possible de se représenter le plan de parcours se déplaçant le long du squelette, en découpant le maillage en tranches.

Nous avons également retenu cet algorithme car il corrigeait les défauts d'autres solutions proposées dans l'état de l'art [2]. Ainsi, il ne demande nativement aucune entrée utilisateur, tel que demandé dans le sujet. De surcroît, son résultat n'est pas aléatoire et doit présenter la même sortie à chaque exécution.

En revanche, cet algorithme a peu de chances de séparer les maillages en trois sous-parties (tige principale, branches et feuilles), et se contentera plus vraisemblablement de deux types de composants (tige principales, opposées aux branches et feuilles). Il serait probablement pertinent de réaliser une analyse supplémentaire sur la segmentation résultante.

Ce projet nous a permis d'avoir une première expérience sur un projet utilisant des maillages en trois dimensions. Cela nous permet de mieux nous rendre compte des problématiques liées au stockage, étudiées en cours, puisque nous avons dû changer de structure au cours de notre projet. De plus, nous avons pu appliquer des connaissances théoriques (géométrie pour la 3D, traitement de maillages) à la pratique, tout en améliorant notre connaissance du C++.
Nous avons également eu l'occasion de réaliser à quel point l'automatisation de la segmentation est intéressante : réalisée manuellement sur un maillage en 3D, elle est arbitraire ; et réaliser le squelette à la main est particulièrement fastidieux.
Enfin, malgré la taille de notre groupe, nous sommes parvenus à diviser les tâches de façon assez satisfaisante, notamment en travaillant sur du pseudo-code avant de passer à l'implémentation elle-même. Sur la dernière itération, nous nous sommes également beaucoup entraidés face au code lui-même.

Nous aurions aimé avoir davantage de temps pour cet exercice ; le projet nécessitait une grande part de travail de recherche et de compréhension, qui aurait pu être réalisée en amont. Ainsi, après avoir choisi notre algorithme, nous avons passé beaucoup de temps à le comprendre, puis à le transformer en pseudo-code. Nous avons ensuite réalisé le code correspondant assez rapidement, sans encore pouvoir le tester puisque les différentes parties de l'algorithme étaient interdépendantes.
En effet, une grande difficulté d'organisation a été la dépendance de la segmentation vis-à-vis du squelette. La création du squelette nous a demandé beaucoup de temps, et la segmentation ne pouvait que peu avancer tant que nous n'avions pas de certitudes quant aux résultats renvoyés par la squelettisation.
La méthodologie agile nous a permis d'être très structurés au départ, lors des deux ou trois premières itérations ; cependant, puisqu'elles étaient très inégales en termes de

temps, la charge de travail s'est retrouvée assez déséquilibrée, et les dernières itérations ont été très importantes. Cette méthodologie ne nous a donc pas permis d'éviter la classique surcharge de travail en fin de projet.

Références :

[1] LI X., TOON T., TAN T., HUANG Z.: Decomposing polygon meshes for interactive applications. In Proceedings of the 2001 symposium on Interactive 3D graphics (2001), pp. 35–42.
[2] *A Survey on Mesh Segmentation Techniques* - Ariel Shamir

Annexes

Exécution manuelle de l'algorithme de squelettisation (pour évaluation)

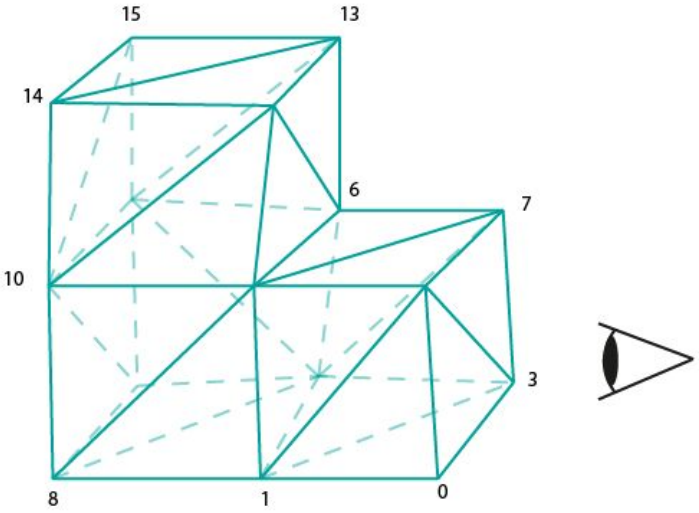

Figure 1 : Maillage complet

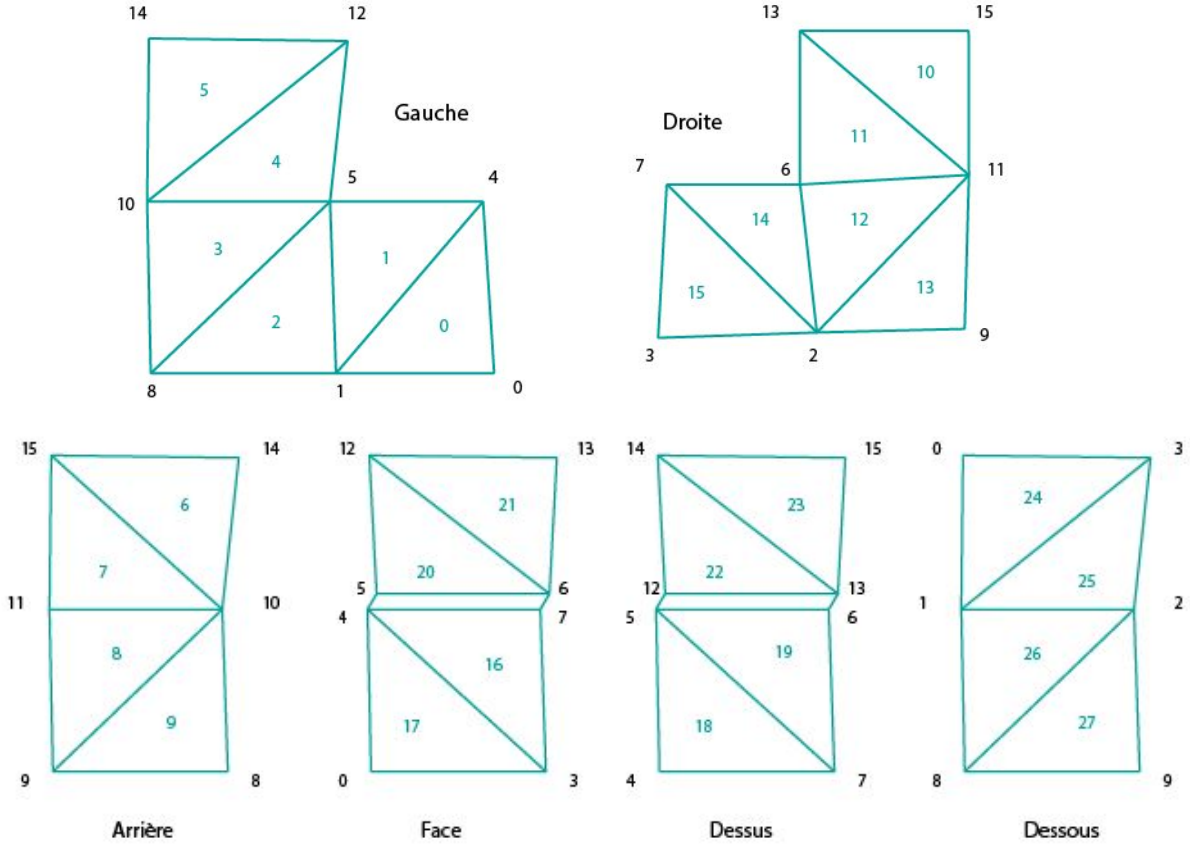

Figure 2 : Détail des différentes vues

Arête (et sommets) à supprimer	Sommet résultant	Faces à stocker dans des ATL	Détail (arête → face stockée)
(10, 11)	16	7 & 8	(15, 16) → 7 (9, 16) → 8
(12, 14)	17	5 & 22	(13, 17) → 22 (10, 17) → 5
(9, 16)	18	9 & 13	(8, 18) → 9 (2, 18) → 13
(0, 4)	19	1 & 18	(1, 19) → 0 (7, 19) → 17
(7, 19)	20	17 & 19	(5, 20) → 18 (3, 20) → 16
(13, 17)	21	22 & 24	(15, 21) → 23 (6, 21) → 21
(2, 3)	22	16 & 26	(1, 22) → 25 (20, 22) → 15
(5, 6)	23	20 & 21	(20, 23) → 19 (21, 23) → 20
(21, 23)	24	5 & 12	(18, 24) → 11 (18, 24) → 4
(1, 8)	25	3 & 27	(22, 25) → 26 (24, 25) → 2
(15, 24)	26	7 & 11	(18, 26) → 6 (18, 26) → 10
(18, 26)	27	4 & 13	(22, 27) → 12 (25, 27) → 3
( )	28	15 & 25	14 24
	29	2 & 28	1 27

Cet exemple 3D a permis de mettre en évidence les cas particulier du prisme et du tétraèdre.

Figure 3 : Squelette de l'objet "Plante4".