Chef d'Oeuvre

Traitement de la géométrie : décomposition de maillages en variétés topologiques

Méthodes et algorithmes

Laura BARROSO
Martin BOUYRIE
Sébastien EGNER

Encadrant : Nicolas MELLADO
Clients : Nicolas MELLADO, Loïc BARTHES

Université Toulouse III - Paul Sabatier
23 novembre 2020

1 Introduction

Ce rapport illustre les méthodes de GUEZIEC et al. [2001] qui permettent la conversion d'une surface **non manifold** en une surface **manifold** sur des topologies existantes. Pour rappel, une surface est **manifold** si chaque arête qui n'est pas un bord partage exactement deux faces et qu'en chaque point du maillage, il existe une sphère de rayon sufisamment petit pour que son intersection avec le maillage soit homothétique à un disque. On dit alors que le **manifold** garantit une bonne représentation de maillage puisqu'il permet alors, d'éviter les artefacts visuellement dérangeants.

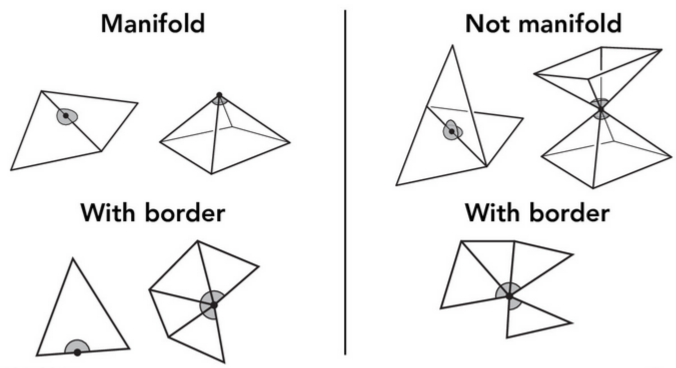


FIGURE 1 – Différence entre une surface **manifold** et une surface **non manifold**. Image de Ng

Ces méthodes ont vu le jour suite à la nécessité de conserver une bonne représentation de maillage. En effet, de nombreuses méthodes et algorithmes nécessitent un maillage **manifold** pour être utilisés. Cependant, de nombreux facteurs peuvent être à l'origine d'une singularité topologique et de ce fait d'un maillage **non manifold**. Parmi ceux-ci, on peut citer par exemple les nombreuses applications de modélisation 3D qui permettent aux artistes de créer des maillages **non manifolds**. Si pour ces artistes un tel maillage ne représente pas de problème, selon la manière dont l'on souhaite utiliser ce maillage par la suite, on peut s'avérer être bloqué.

Beaucoup d'articles scientifiques se concentrent sur le problème de surfaces **non manifolds** puisque c'est un problème dominant dans les représentations de maillage. Certaines méthodes plus anciennes se concentrent sur l'efficacité et l'optimisation des réparations de ces maillages particuliers.

C'est le cas de l'article de Jarek ROSSIGNAC [2000], qui propose une méthode permettant de réduire considérablement le nombre de sommets coïncidents en évitant les duplications de sommets inutiles pour réduire drastiquement leur coût de stockage. D'autres articles, comme celui de FLORIANI et al. [2003], plus récent que l'article que nous présentons ici, proposent une réflexion sur une correction d'objets **non manifold** pour n'importe quelle dimension, avec des méthodes dont l'efficacité ne varie pas selon le nombre de singularités présentes. Cet article reproche notamment aux articles existants de se focaliser sur des réparations de maillages **non manifolds** par des méthodes 2D.

Certains articles se penchent davantage sur la question du niveau de modification, c'est à dire, à la modification de la connectivité et/ou de la géométrie que cela engendre. C'est le cas de l'article de ATTENE et al. [2009] qui s'y interroge localement. D'autres encore recensent les articles existants en catégorie et permettent une meilleure visualisation. L'article de ATTENE et al. [2013] permet ainsi de mieux appréhender les différents avantages et inconvénients que chaque méthode peut procurer et de comparer ces méthodes selon leurs propriétés et leurs garanties pour une meilleure application.

Nous présentons ici les méthodes de *cutting* et de *stitching* qui permettent de modifier n'importe quelle surface d'entrée en une surface **manifold**, en décrivant leur fonctionnement et en les expliquant via leur pseudo-code.

2 Explication des méthodes

L'algorithme général de l'article se compose de deux phases principales : tout d'abord le *cutting*, qui va permettre de découper la surface d'entrée en plusieurs surfaces **manifold** distinctes et ensuite la phase de *stitching* (optionnelle), qui va s'occuper de *recoudre* ces différentes surfaces **manifold** afin d'obtenir notre surface originale sous la forme d'une surface **manifold**.

Avant de pouvoir appliquer la phase de *cutting*, il nous faut être sûr que la surface ne comporte pas de **degenerate faces**. C'est évoqué très brièvement dans l'article et si l'on peut imaginer cela comme un prérequis dont l'utilisateur s'est lui-même assuré, nous avons décidé de rajouter explicitement une phase durant laquelle ces éventuelles **degenerate faces** seraient supprimées de la surface. Cela assure que la suite de l'algorithme ne posera pas de problème.

La structure générale de l'algorithme ressemble donc à ceci :

Algorithme 1: Algorithme complet

Entrées: Une surface abstraite polygonale, définie par ses sommets et faces.

Sorties: Un ensemble de surfaces manifold ou une seule surface manifold (selon si l'on applique l'étape optionnelle de *stitching* ou non).

eliminate_degenerate_faces()
cutting()
stitching() `// Optionnel`

2.1 Élimination des *degenerate faces*

L'élimination des **degenerate faces** est relativement simple. Une **degenerate face** étant définie par la présence d'au moins deux sommets identiques pour une même face, il suffit d'itérer sur toutes les faces de la surface et de vérifier qu'aucune d'entre elles ne possèdent une telle propriété, auquel cas on la supprime tout simplement.

L'algorithme se présente donc comme cela :

Algorithme 2: Suppression des **degenerate faces**

Entrées: Une surface abstraite polygonale, définie par ses sommets et faces.

Sorties: Cette même surface, sans les **degenerate faces**.

Pour chaque face f de la surface **faire**
- *sommets* $\longleftarrow$ []
- **Pour chaque** sommet s lié à la face f **faire**
  - **Si** s est dans *sommets* **alors**
    - Retirer f de la surface (ainsi que les arêtes et sommets liés)
  - **Sinon**
    - Ajouter s à *sommets*
  - **Fin**
- **Fin**

Fin

Cet algorithme a une complexité linéaire, corrélée au nombre de faces de la surface d'entrée.

2.2 *Cutting*

Le *cutting* est l'étape au cours de laquelle la surface d'entrée, possiblement une surface **non manifold**, est *découpée* le long d'arêtes et de sommets singuliers. De ce découpage résulte un certain nombre de surfaces **manifolds**. Il existe deux approches pour cette étape de *cutting* : l'approche *globale*, et l'approche dite *locale*.

2.2.1 Identification des arêtes singulières

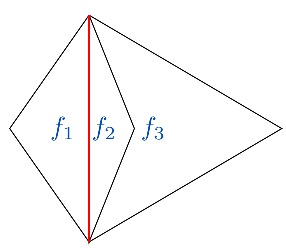


Figure 2 – Un exemple d'**arête singulière** : ici, l'arête rouge est liée à 3 faces (f_1, f_2 et f_3). Cela fait d'elle une **arête singulière**.

Quelle que soit l'approche choisie, on commence par identifier les **arêtes singulières** de notre surface d'entrée. Une **arête singulière** étant définie par la présence d'au moins trois faces incidentes, il nous suffit d'identifier parmi toutes les arêtes de notre surface celles qui présentent une telle propriété. Lorsque des arêtes pareilles sont rencontrées, il suffit de les marquer comme étant *singulières*. Ce marquage nous servira dans la suite de la phase de *cutting*.

Algorithme 3: Identification des **arêtes singulières**

Entrées: Une surface abstraite polygonale, définie par ses sommets et faces.

Sorties: La liste des **arêtes singulières** présentes dans cette surface.

```
aretes_singulieres ⟵ []
Pour chaque face f de notre surface faire
    Pour chaque arête a incidente à f faire
        Si a n'est pas marqué comme étant singulière et que a est liée à
          au moins 3 faces différentes alors
            Marquer a comme étant singulière
            Rajouter a à aretes_singulieres
        Fin
    Fin
Fin
```

Là encore, la complexité algorithmique de cette identification est linéaire, dépendant du nombre de faces de la surface d'entrée.

2.2.2 Approche locale

Le *cutting* local est préféré au *cutting* global dans le cas où la surface d'entrée possède un nombre réduit de **singularités topologiques**. En effet, la complexité de l'approche locale est proportionnelle au nombre de **sommets singuliers** de la surface d'entrée, multiplié par la plus grande valence d'un **sommet singulier**.

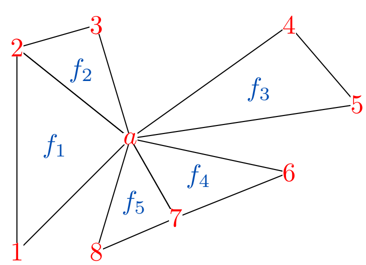


Figure 3 – Un exemple de **sommet singulier isolé** : Le sommet a est un **sommet singulier isolé** : en effet, $l(a)$ n'est ni une *chaîne*, ni un *cycle* et les quatres arêtes liées à a ne sont pas des **arêtes singulières**

La méthode locale requiert une phase d'identification supplémentaire, celle des **sommets singuliers isolés**. Un **sommet singulier isolé** correspond à un **sommet singulier** qui n'est lié à aucune **arête singulière**. Pour illustrer sa définition, prenons la figure 2. Aucune arête liée à a (parmi les arêtes formées par les sommets $(1, a), (2, a), (3, a)$ et $(4, a)$) ne sont des **arêtes singulières**. De plus, si l'on construit $l(a)$, le graphe dont les noeuds sont les faces du sous ensemble a*, on peut voir dans la figure 4, que ce graphe n'est ni une *chaîne*, ni un *cycle*. On en déduit donc, qu'il s'agit d'un **sommet singulier isolé**.

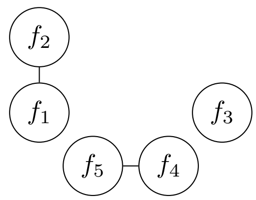


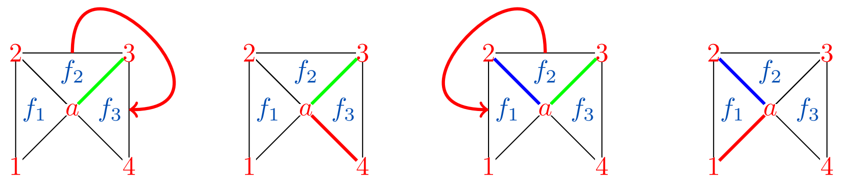
Figure 4 – Le lien du sommet a, également noté $l(a)$, avec a le sommet de la figure 3

Pour réaliser une telle identification, on itère sur tous les sommets a non liés à une **arête singulière** (celles-ci ayant été identifiées précedemment, il n'est pas difficile d'itérer ainsi). Pour chacun de ces sommets, il ne nous reste alors plus qu'à regarder si $l(a)$ constitue une *chaîne* ou un *cycle*. Pour cela, on va partir d'une face arbitraire de a^* et se déplacer, à partir de cette première face, sur ses faces adjacentes (les faces qui partagent au moins une arête commune) et incidentes sur a, dans les deux sens. Si a correspond à un **sommet singulier isolé**, alors il n'est pas possible de parcourir toutes les faces incidentes a en se déplaçant de cette manière, quelle que soit la face initiale choisie.

Le fonctionnement de ce *déplacement* autour des différentes faces adjacentes de a est illustré dans la figure 5 et correspond à la partie de l'algorithme 4 allant de la ligne **2** à la ligne **17**.

Quelques applications de cette méthode de déplacement sont illustrées dans la figure 6.

Si l'on arrive à se déplacer sur toutes les faces adjacentes de a de cette manière, alors on en conclut que a n'est pas un **sommet singulier isolé**. Dans le cas contraire, a correspond à un **sommet singulier isolé**. Cette conclusion correspond aux lignes **18**, **19** et **20** de l'algorithme 4.



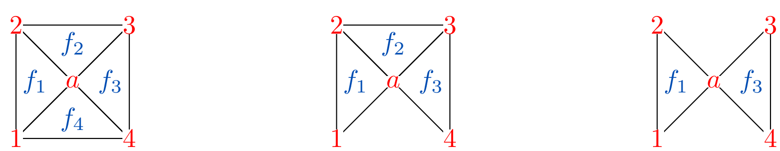
(a) Soit f_2 la face initiale choisie arbitrairement. On choisit ensuite, là encore de manière arbitraire, une arête liée à f_2 et incidente sur a. Ici, cela correspond à l'arête verte. On récupère ensuite la seconde face qui partage cette arête avec f_2, ici f_3.

(b) On procède de la même manière avec f_3 maintenant : on choisit une arête liée à f_3, incidente sur a et différente de l'arête verte précédemment trouvée. Ici, cela correspond à l'arête rouge. Cependant, c'est une **arête frontalière**, on ne peut donc pas continuer dans ce sens-là.

(c) On repart de la face initiale f_2. On choisit une arête liée à f_2, incidente sur a et différente de l'arête verte. Ici, cela correspond à l'arête bleue. On récupère ensuite la seconde face qui partage cette arête, ici f_1.

(d) Comme à la seconde étape, on identifie une arêté liée à f_1, incidente sur a et différente de l'arête bleue. Ici, cela correspond à l'arête rouge. Cependant, c'est une **arête frontalière**, on ne peut donc pas continuer dans ce sens-là. On s'arrête alors, on a exploré en tout **3** faces, ce qui correspond également à la valence de a : a n'est donc pas un **sommet singulier isolé**.

Figure 5 – Le déplacement utilisé pour l'identification des **sommets singuliers isolés**.



(a) Ici, quelle que soit la face initiale choisie, on peut se *déplacer* sur toutes les autres faces incidentes sur a. a n'est donc pas un **sommet singulier isolé**.

(b) Là aussi, quelle que soit la face initiale choisie, on peut se *déplacer* sur toutes les autres faces incidentes sur a. Étant donné que a est lié à deux **arêtes frontalières**, il se peut que l'on doive *changer de sens* en repartant de la face initiale pour toutes les atteindre cependant. a n'est donc pas un **sommet singulier isolé**.

(c) Avec ce maillage, il est impossible de se *déplacer* sur les deux faces incidentes de a, quelle que soit la face initiale choisie. a est donc un **sommet singulier isolé**.

Figure 6 – Quelques exemples d'identification de **sommets singuliers isolés**. Pour ces trois maillages, a n'est lié à aucune **arête singulière**.

Algorithme 4: Identification des **sommets singuliers isolés**

Entrées: Une surface abstraite polygonale, définie par ses sommets et faces.

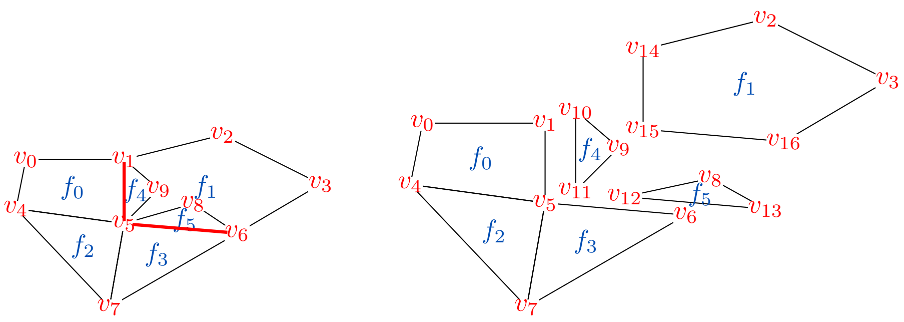
Sorties: La liste des **sommets singuliers isolés** présents dans cette surface.

Pour chaque sommet v qui n'est pas lié à une **arête singulière faire**
- On choisit de manière arbitraire une face initiale f, incidente à v
- On récupère de manière arbitraire une arête e_1, incidente à f et à v
- $nb\_faces\_explorees \longleftarrow 0$
- `// On pivote`
- **faire**
  - On récupère une face g, qui partage e_1
  - On récupère une arête a, liée à g, incidente à v et telle que $a \neq e_1$
  - $nb\_faces\_explorees \longleftarrow nb\_faces\_explorees + 1$
- **tant que** $f \neq g$ **ET** a n'est pas une **arête frontalière**
- **Si** a est une **arête frontalière alors**
  - On récupère une arête e_2, liée à f, incidente à v et telle que $e1 \neq e2$
  - `// On pivote dans l'autre direction`
  - **faire**
    - On récupère une face g, qui partage e_2
    - On récupère une arête a, liée à g, incidente à v et telle que $a \neq e_2$
    - $nb\_faces\_explorees \longleftarrow nb\_faces\_explorees + 1$
  - **tant que** $f \neq g$ **ET** g n'est pas une **arête frontalière**
- **Fin**
- **Si** $nb\_faces\_explorees \neq \deg(v)$ **alors**
  - Marquer v comme étant un **sommet singulier isolé**
- **Fin**

Fin

Une fois l'identification des **sommets singuliers isolés** réalisée, on peut passer au *découpage* de notre surface. La *découpe* se fait sur chaque sommet lié à une **arête singulière** et chaque **sommet singulier isolé**. Grâce aux identifications précédemment réalisées, on peut facilement récupérer ces sommets et ainsi procéder à la *découpe*.

Pour réaliser la *découpe*, on itère sur chacun des sommets à découper v_i, et l'on partitionne les faces de v_i^* en plusieurs sous-ensembles de **faces atteignables**. Deux faces sont dites *atteignables* si elles partagent un **sommet régulier** et incident sur v_i. On *multiplie* ensuite v_i par le nombre de sous-ensembles ainsi créés. *Multiplier* un sommet correspond simplement à en créer une copie, avec les mêmes coordonnées et propriétés. Une fois toutes les copies créées, on relie ces nouveaux sommets à leurs sous-ensembles respectifs. Cette procédure est illustrée dans la figure 7.



(a) Surface originale : les **arêtes singulières** sont affichées en rouge. Les sommets v_1, v_5 et v_6 sont marqués comme étant à *découper*.

(b) Ici, on copie le sommet $v1$ deux fois : on obtient v_{10} et v_{14}. On copie le sommet v_6 deux fois également : on obtient v_{13} et v_{16}. On recopie cependant v_5 trois fois : on obtient v_{11}, v_{12} et v_{15}. Il faut bien comprendre que les coordonnées de ces copies sont inchangées par rapport à leur sommet original. Si elles sont affichées à différents endroits sur cette figure, c'est simplement par un souci de rendre l'explication plus claire. Une fois toutes ces copies créées, on les relie à leur sous-ensemble associé. On vient de réaliser le *cutting* local et l'on obtient 3 surfaces **manifold** comme résultat.

Figure 7 – Illustration du *cutting* local.

L'algorithme final pour le *cutting* local ressemble à cela :

Algorithme 5: *Cutting* local

Entrées: Une surface abstraite polygonale, définie par ses sommets et faces, ainsi que la liste des **arêtes singulières** et celle des **sommets singuliers isolés**, identifiés précédemment.

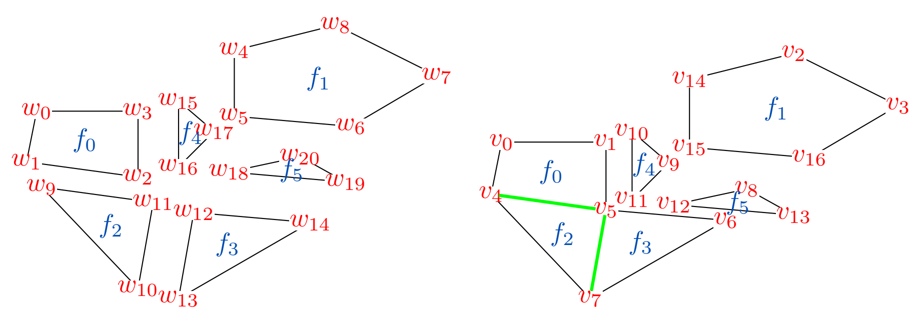
Sorties: Une liste de surfaces **manifold**.

```
Pour chaque arête marquée singulière a de la surface faire
    Marquer les deux sommets incidents de a comme étant des sommets
      à découper
Fin
Pour chaque sommet singulier isolé s faire
    Marquer s comme étant un sommet à découper
Fin
Pour chaque sommet marqué à découper v_i faire
    Partitionner les faces de v_i^* en plusieurs sous-ensembles de faces
      dites atteignables
    On note n_c le nombre de sous-ensembles ainsi créés
    On crée n_c − 1 copie(s) de v_i, avec les mêmes coordonnées et
      propriétés, et on labellise les différentes instances de v_i de 0 à n_c − 1
    Pour chaque face f dans v_i^* faire
        Relier la face à la copie de v_i correspondante à son sous-ensemble
    Fin
Fin
```

2.2.3 Approche globale

L'approche globale est préférée à l'approche locale lorsque la surface d'entrée présente beaucoup de **singularités topologiques**. En effet, la complexité de cette approche dépend en partie du nombre d'**arêtes régulières** de la surface d'entrée.

L'approche globale commence par la création d'une nouvelle surface, nommée Σ, qui correspond à la surface de départ S pour laquelle on *découpe* chaque arête (de la même manière que pour le *cutting* local). Ensuite, en utilisant les **arêtes singulières** précédemment identifiées, on peut regrouper certaines faces pour obtenir les sous-ensembles attendus. En effet, on peut regrouper toutes les faces qui partagent entre elles une **arête régulière** (une **arête régulière** étant définie comme étant le contraire d'une **arête singulière**, l'identification de ces dernières va nous permettre de facilement effectuer ces regroupements). Le fonctionnement du *cutting* global est illustré dans la figure 8.



(a) La surface Σ, obtenue après avoir *découpé* autour chaque face de la surface S d'origine.

(b) On *regroupe* toutes les faces qui partagent une **arête régulière**. Ici, on peut regrouper les faces f_0, f_2 et f_3 car les arêtes affichées en vert sont *régulières*. Là encore, comme pour le *cutting* local, les coordonnées de sommets v_1, v_{10} et v_{14} sont similaires et si elles sont affichées à différents endroits sur cette figure, c'est simplement par un souci de rendre l'explication plus claire. Il en est de même pour les sommets v_5, v_{11}, v_{12} et v_{15} ainsi que les sommets v_6, v_{13} et v_{16}. On vient de réaliser le *cutting* global et l'on obtient 3 surfaces **manifold** comme résultat.

FIGURE 8 – Illustration du *cutting* global.

L'algorithme final pour le *cutting* global ressemble à cela :

Algorithme 6: *Cutting* global

Entrées: Une surface S, abstraite polygonale, définie par ses sommets et faces, ainsi que la liste des **arêtes singulières**, identifiées précédemment.

Sorties: Une liste de surfaces **manifold**.

Pour chaque arête marquée singulière a de la surface **faire**
  Marquer les deux sommets incidents de a comme étant des sommets à découper
Fin
$\Sigma \longleftarrow S$
Pour chaque sommet marqué à découper v_i **faire**
  $liste\_faces \longleftarrow []$
  Pour chaque arête non frontalière a incidente sur v_i **faire**
    Pour chaque face f incidente sur a **faire**
      Si f n'est pas dans $liste\_faces$ **alors**
        Rajouter f dans $liste\_faces$ en effectuant les copies des sommets nécessaires
      Fin
    Fin
  Fin
  Pour chaque arête régulière a incidente sur v_i **faire**
    Regrouper les deux faces incidentes sur a en fusionnant les extrémités de a
  Fin
Fin

2.3 Stitching

Pour *recoudre* le maillage, on distingue deux stratégies : le *pinching* et le *snapping*. Avant d'appliquer ces méthodes, on doit construire les **frontières** d'arêtes **stitchable**. Il est important de souligner que la notion de **stitchable** est différente selon la stratégie appliquée. Chaque notion de **stitchable** est détaillée dans les sous-parties associées.

Algorithme 7: Construction des frontières

Entrées: Une surface abstraite polygonale, définie par ses sommets et faces.

Sorties: Liste des frontières

```
liste_frontieres ⟵ []
Pour chaque arête stitchable a faire
    Si a est adjacente à une frontière existante alors
        Ajouter a à la frontière
    Sinon
        Ajouter une nouvelle frontière à liste_frontieres
        Ajouter a à cette frontière
    Fin
Fin
```

2.3.1 Pinching

Le but de cette stratégie est de *recoudre* uniquemment les arêtes créées pendant l'étape de cutting. Pour cette stratégie, une arête est donc **stitchable** uniquement si elle a été coupée pendant l'opération de *cutting*. Cette stratégie referme les frontières sur elles-mêmes et garantit que les *coutures* ne créent de nouvelles singularités.

Ainsi, on va chercher dans chaque **frontière**, des paire d'arêtes adjacentes qui partagent un sommet d'une **frontière** à l'autre. Ce sommet est appelé le pivot, c'est autour de lui que la couture prend forme. Une fois la couture effectuée, on se déplace le long de la **frontière** jusqu'à trouver de nouvelles arêtes à coudre.

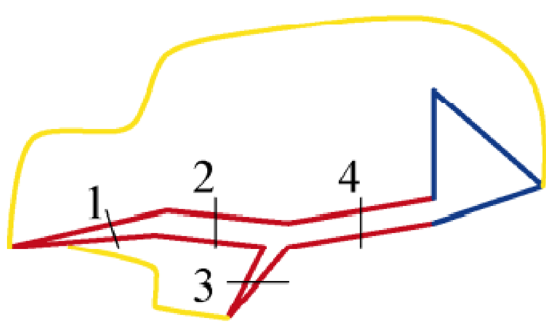


Figure 9 – Illustration de la stratégie *pinching*

Sur la figure 9, nous pouvons observer les différentes paires d'arêtes adjacentes qui sont représentées de couleur rouge et qui sont donc **stitchable**. Dans cet exemple, nous allons recoudre 4 paires. La première paire représentée par le chiffre 1, a bien un sommet commun situé sur la gauche. Nous pouvons donc recoudre les deux arêtes de la paire 1. Après cette réalisation, la paire 2 adjacente à la paire 1, possède alors le nouveau sommet formé par la paire 1 qui s'est alors déjà recousu. La paire 2 peut à son tour être recousue. Ce phénomène se répète donc jusqu'au contre exemple représenté de couleur bleu.

Algorithme 8: Pinching

Entrées: Une surface abstraite polygonale passée par l'étape de *cutting*.
Sorties: La surface d'entrée *recousue* et manifold

Pour chaque frontière f de notre surface **faire**
  Choisir une paire d'arêtes adjacentes (v_0, v_1) et (v_0, v_2) dans f
  faire
    Pinch (v_0, v_1) et (v_0, v_2)
    Choisir 2 nouvelles arêtes adjacentes de f
  tant que f contient des arêtes
Fin

2.3.2 Snapping

Contrairement à la stratégie de *pinching*, celle du *snapping* peut s'appliquer sur des arêtes qui n'ont pas été créées par l'étape de *cutting*. Une arête sera **stitchable** si elle appartient à une **frontière**. Une couture sera envisagée lorsque deux arêtes **stitchable** sont suffisamment proches, c'est-à-dire si leurs sommets respectifs se trouvent à une certaine distance.

La priorité de cette stratégie est de *recoudre* les **frontières** qui sont suffisamment proches. Puisque cette stratégie recherche des arêtes **stitchable** localement proches, on peut réduire la complexité de recherche avec un octree. Cette méthode, proposée dans l'article de GUEZIEC et al. [2001], s'exécute en deux passes : la première consiste à cibler les paires dont les arêtes n'appartiennent pas à la même frontière tandis que la seconde s'occupe des arêtes restantes.

Quand on trouve une paire d'arêtes éligible, on va dans un premier temps vérifier que le point de couture entre ces arêtes est valide. Considérons deux arêtes (v_0, v_1) et (v'_1, v'_0) éligibles. Le point de couture est valide s'il vérifie les trois conditions suivantes :

— v_0^* et v_1^* n'ont pas d'intersection avec $v_1'^*$ et $v_0'^*$
— aucune des arêtes (v_0, v'_1) ou (v_1, v'_0) n'est une arête **stitchable**
— soit v un sommet du maillage, il n'existe pas d'arête de la forme $(v,\, v_0)$ et $(v,\, v'_1)$ ou de la forme $(v,\, v_1)$ et $(v,\, v'_0)$

Si la paire n'est pas valide, elle est rejetée et on passe à la suivante. Quand on trouve une paire valide, on veut vérifier que le point de couture ne va pas créer de singularité. Pour cela, on étudie la topologie des sommets concernés et notamment leur **étoile**. Une singularité est créée lorsque les sommets concernés sont reliés par une arête qui n'est pas une **arête frontalière**.

La structure d'une passe de snapping est la suivante.

Algorithme 9: Snapping

Entrées: Une surface abstraite polygonale passée par l'étape de *cutting*.
Sorties: Liste des frontières

Construire l'octree
Pour chaque feuille f de l'octree **faire**
    Pour chaque paire d'arêtes $(v_0,\ v_1)$ et $(v'_1,\ v'_0)$ de f **faire**
        Si (v_0, v_1) et (v'_1, v'_0) sont suffisamment proches **alors**
            `// Test de la validité de la couture`
            $valide = (v_0^*$ **ET** v_1^* n'ont pas d'intersection avec $v_1'^*$ **ET** $v_0'^*)$
            $valide = valide$ **ET** (aucune des arêtes (v_0, v'_1) ou (v_1, v'_0) n'est une arête **stitchable**)
            $valide = valide$ **ET** (il n'existe pas d'arête de la forme $(v,\ v_0)$ et $(v,\ v'_1)$ ou de la forme $(v,\ v_1)$ **ET** $(v,\ v'_0)$)
            Si *valide* **alors**
                // vérifier le résultat de la couture
                Si la couture entre v_0 et v'_1 ne crée pas d'**arête singulière** et génère un stitch implicite **alors**
                    Merge $v_0\ v'_1$
                Fin
                Si la couture entre v_1 et v'_0 ne crée pas d'**arête singulière** et génère un stitch implicite **alors**
                    Merge $v_1\ v'_0$
                Fin
                Si il n'y a pas eu de couture et qu'aucun des deux ne crée d'**arête singulière alors**
                    Merge $v_0\ v'_1$
                    Merge $v_1\ v'_0$
                Fin
                Si une des deux coutures a été effectuée et que l'autre ne génère pas d'**arête singulière alors**
                    Merge celle qui ne l'est pas déja
                Fin
            Fin
        Fin
    Fin
Fin

Le *merge* entre deux sommets traduit la couture entre ceux-ci, c'est une couture explicite (*explicit stitch*). Cependant, celles-ci peuvent générer des coutures au préalable appelée couture implicite (*implicit stitch*). Cette opération *merge* n'étant pas plus détaillée dans l'article étudié, il est difficile de comprendre son fonctionnement précisément.

3 Organisation du projet

Nous avons décidé d'utiliser *Discord* pour la gestion et communication interne du projet. L'application a l'avantage de rassembler des canaux vocaux pour les réunions et textuels pour des échanges asynchrones. Étant donné le contexte actuel de la crise sanitaire de la Covid-19, les rendez-vous avec l'encadrant ont lieu de préférence à distance par *Discord* ou *Zoom*.

Concernant notre plan de réalisation, puisque les sous-parties de l'algorithme *cutting* et *stitching* proposent chacune deux méthodes différentes, nous pensions commencer à implémenter une des stratégies de cutting et une des stratégies de stitching. Ce choix nous permettra d'avoir un résultat fonctionnel dans le cas d'un imprévu qui nous empêcherait de terminer le travail dans sa totalité. Nous avons donc décidé d'implémenter dans un premier temps la méthode globale de *cutting* qui permettra de réaliser des tests en faisant varier le nombre de **singularités topologiques**, puis la méthode de *pinching*.

Concernant les différentes technologies que nous allons utiliser pour ce chef d'oeuvre, on peut tout d'abord parler de *Radium Engine*, un moteur de rendu mis en place par l'équipe STORM de l'IRIT. Nous utiliserons le gestionnaire de version *GitLab* pour notre implémentation de l'algorithme. Concernant le site Web présentant le Chef d'Oeuvre, nous avons pensé à utiliser *Github*, notamment car *Github Pages* nous permettrait de réaliser ce site simplement et efficacement. Nous utiliserons également la bibliothèque *OpenMesh* (sur laquelle *Radium Engine* repose d'ailleurs). Enfin, nous avons envisagé d'utiliser différents outils pour faciliter le développement du projet tel que *Doxygen* pour la documentation, *TravisCI* pour l'intégration continue et *Catch2* pour la génération de tests unitaires et éventuellement un outil de formatage comme *clang-format*.

Avec les différentes listes que l'on doit utiliser dans les méthodes expliquées, on peut notamment penser aux listes d'arêtes et sommets singuliers, l'optimisation du parcours de ces listes est un point crucial de l'implémentation. Puisque nous avons déjà prévu d'utiliser la bibliothèque *OpenMesh* et que celle-ci offre la possibilité de stocker des informations supplémentaires sur les éléments d'un maillage, il nous semble intéressant de l'utiliser pour gérer l'efficacité de ces listes.

Concernant les structures de données complexes auxquelles nous serons confrontés durant ce chef d'oeuvre, on peut évoquer la structure du type octree qu'utilise la méthode de *snapping* et dont l'implémentation existe déja en abondance sur de nombreux projets open source. Autrement, les différentes surfaces, faces, arêtes et sommets manipulés au cours des méthodes présentées pourront être représentées à l'aide des structures proposées par la bibliothèque *OpenMesh*.

Glossaire

arête frontalière Une arête est dite *frontalière* si elle ne possède qu'une seule face incidente. 6, 7, 12

arête non frontalière Une arête est dite *non frontalière* si au moins deux faces lui sont incidentes.. 10

arête régulière Une arête est dite régulière si au plus deux faces lui sont incidentes. 9, 10, 15

arête singulière Une arête est dite *singulière* si au moins trois faces lui sont incidentes. Autrement, on parle d'arête *régulière*. 3–10, 13, 15

degenerate face On parle de *degenerate face* lorsqu'une face possède au moins deux sommets identiques. 2, 3

face atteignable Deux faces sont dites *atteignables* par rapport à un sommet v si elles partagent une **arête régulière** et incidente sur le sommet v.. 7

frontière Une *frontière* est un ensemble d'arêtes frontalières connectées. 10–12

manifold Une surface polygonale est dite *manifold* si tous ses sommets sont réguliers. Dans l'autre cas, on parle d'une surface *non manifold*. 1–3, 8–10

non manifold Une surface polygonale dite *non manifold* si au moins un de ses sommets est singulier. Dans l'autre cas, on parle d'une surface *manifold*. 1–3

singularité topologique Une singularité topologique correspond soit à un **sommet singulier**, soit à une **arête singulière**. 4, 9, 14

sommet régulier Un sommet v est dit *régulier* si $l(v)$ est une *chaîne* ou un *cycle*. Autrement, on parle de sommet *singulier*. 7

sommet singulier Un sommet v est dit *singulier* si $l(v)$ n'est ni *chaîne*, ni un *cycle*. Autrement, on parle de sommet *régulier*. 4, 15

sommet singulier isolé Un sommet *singulier isolé* correspond à un **sommet singulier** qui n'est pas une extrémité d'une **arête singulière**. 4–8

stitchable Une arête est dite *stitchable* si elle est éligible pour être recousue. 10–13

étoile L'*étoile* d'un sommet est l'ensemble des faces dont il fait partie. 12

Références

André GUEZIEC, Gabriel TAUBIN, Francis LAZARUS, and Bill HORN. Cutting and stitching : Converting sets of polygons to manifold surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 7(2), 2001.

Ren Ng. URL `https://cs184.eecs.berkeley.edu/sp19/lecture/8-17/meshes-and-geometry-processing`.

David CARDOZE Jarek ROSSIGNAC. Matchmaker : Manifold breps for non-manifold r-sets. *Georgia Institute of Technology*, 2000.

Leila De FLORIANI, Franco MORANDO, and Enrico PUPPO. Representation of non-manifold objects through decomposition into nearly manifod parts. *Department of Computer and Information Sciences, Università di Genova, Via Dodecaneso 35, 16146 Genova - Italy*, 2003.

Marco ATTENE, Daniela GIORGI, Massimo FERRI, and Bianca FALCIDIENO. On converting sets of tetrahedra to combinatorial and pl manifolds. *Institute of Applied Mathematics and Information Technology - National Research Council, Genoa, Italy - Department of Mathematics - Bologna University, Bologna, Italy*, 2009.

Marco ATTENE, Marcel CAMPEN, and Leif KOBBELT. Polygon mesh repairing : An application perspective. *IMATI-GE Consiglio Nazionale delle Ricerche and RWTH Aachen University*, 2013.