

UNIVERSITÉ AIX-MARSEILLE,  
FACULTÉ DES SCIENCES : LUMINY



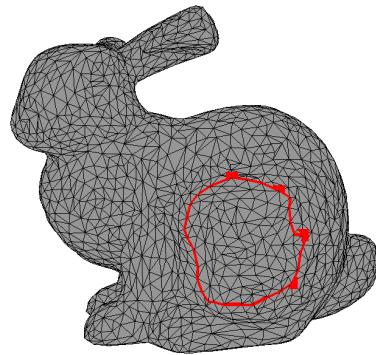
MASTER INFORMATIQUE  
PARCOURS GÉOMÉTRIE ET INFORMATIQUE GRAPHIQUE

---

Projet de Fin d'Études :

***Segmentation 3D par la méthode du Livewire***

---



Auteurs :

Erwan LERIA  
Johan MABILY  
Pierre MATTIOLI

Encadrant :

Romain RAFFIN

# Segmentation 3D par la méthode du Livewire

Erwan Leria<sup>1</sup>, Johan Mably<sup>1</sup>, Pierre Mattioli<sup>1</sup>

<sup>1</sup>Aix-Marseille Université

## Résumé

Ce rapport traite de la mise en oeuvre d'une méthode de livewire 3D pour de la segmentation sur des maillages. La méthode produite s'inspire de méthodes optimisées pour une interaction avec un utilisateur, mais ici on cherche à produire un livewire adapté pour traiter des contours/tracés déjà produits. Ces contours obtenus grâce à un algorithme de machine learning, peuvent être complexes. C'est pourquoi nous essayons de remanier le livewire 3D afin de détecter au mieux les parties d'un maillage qui correspondent à ces contours.

**Mots clé :** Modélisation géométrique, nuage de points, maillages, contour, segmentation, Livewire

## 1. Introduction

Dans le cadre de notre seconde année de master informatique option « géométrie et informatique graphique - GIG », il nous est proposé un projet de fin d'études. Ce projet a pour but de nous permettre de mettre en pratique nos connaissances et nos compétences professionnelles, ayant pour finalité le développement et l'analyse d'algorithmes en accords avec nos intérêts professionnels et la spécialité de notre master. La segmentation 3D est un domaine que nous ne connaissons pas particulièrement. C'est pourquoi nous étions curieux d'en apprendre plus sur ces méthodes. C'est donc dans cet intérêt que nous avons choisi ce projet, dont l'intitulé est la « Segmentation 3D par la méthode du Livewire ».

En général, la méthode du Livewire, référencée également comme « Intelligent Scissors » est largement utilisée en 2D [Wik] pour déterminer des zones, dont les contours sont adaptés au fur et à mesure que l'utilisateur trace le contour. Nous voulons adapter cette méthode à la sélection de zones qui peuvent nous intéresser sur un maillage 3D [Bag07] [Gra06] [FM06], en fonction de critères plus larges (angles dièdres, courbures, orientation des normales...). Nous nous appuierons donc sur l'algorithme du Livewire 2D et, pour passer à une segmentation en 3D, nous implémenterons la méthode du Livewire employée dans la thèse réalisé par William Kiefer [Kie04].

Dans un premier temps, il faut tout de même savoir que la méthode utilisée dans cette thèse [Kie04], s'applique pour une interaction avec un utilisateur capable de choisir les parties d'un maillage à segmenter. Un premier travail sera donc d'adapter la méthode, pour recevoir en entrée des contours déjà tracés. Une des raisons qui motive

ce choix, est de pouvoir utiliser de la segmentation couplée à de l'intelligence artificielle. En effet, il existe des algorithmes nous permettant de pouvoir caractériser des parties d'images ou du nuage de points via le Machine Learning. Nous utiliserons d'ailleurs des données issues d'une de ces méthodes [MLT19]. Cette façon de faire peut alors se révéler très pratique, notamment pour des données de maillages obtenus par photogrammétrie, et dispense d'une interaction avec un utilisateur.

Mais une des complications existantes dans la méthode d'identification que l'on utilise, est qu'elle ne peut s'utiliser que sur des nuages de points. Nous devrons donc trouver une méthode pour projeter sur le maillage, les contours identifiés sur le nuage de point.

Afin de mener à bien le projet, nous effectuerons plusieurs étapes. Dans un premier temps, nous présenterons plus en détails la méthode du livewire. Ensuite, nous expliquerons notre méthode d'extraction et de projection de contours d'un nuage de point à un maillage. Puis nous exposerons les tests et les résultats obtenus avec nos méthodes, en commençant par des tests simples pour valider l'utilisation de notre livewire, et en terminant par des résultats obtenus avec nos méthodes de projection sur des données complexes. Pour finir nous parlerons des possibles améliorations et travaux futurs pouvant être apportés à notre méthode.

## 2. Livewire

Notre but est donc d'implémenter au mieux la méthode décrite dans la thèse « Intelligent Scissoring for Interactive Segmentation of 3DMeshes » réalisé par William Kiefer [Kie04]. Toutefois, cette méthode est initialement optimisée pour une interaction avec un utilisateur devant tracer à la main un contour sur un maillage. Nous ne nous

concentrerons donc pas sur cette partie puisqu'elle ne sera pas prise en compte dans notre projet. Bien sûr nous apporterons des modifications et ajustements pour mieux correspondre à notre approche et aux résultats souhaités. Par ailleurs et afin de faire nos premiers tests, nous sélectionnerons tout de même à la main des sommets représentant des parties de contour.

Avant d'entrer le vif du sujet il a d'abord fallu nous intéresser à la méthode du livewire en 2D. C'est ensuite qu'il nous sera possible de l'adapter à notre problème 3D.

## 2.1. Livewire 2D

Le Livewire 2D a pour but de pouvoir choisir une partie intéressante à extraire d'une image 2D. Pour effectuer cela, l'utilisateur définit le point de départ en cliquant sur le pixel de l'image 2D, appelé ancre. Puis, alors qu'il commence à déplacer la souris sur d'autres points, le chemin de coût minimum est tracé de l'ancre au pixel où se trouve la souris, se changeant si l'utilisateur déplace la souris. Le coût d'un chemin est calculé en fonction de chaque pixel passant par celui-ci. Il existe plusieurs méthodes pour obtenir le coût d'un pixel mais la plus importante reste l'amplitude du gradient. Le coût du pixel nous permettra de savoir la direction qui doit être prise (haut, bas, gauche, droite) pour construire le chemin le plus optimal. S'il veut choisir le chemin qui est affiché, il clique simplement à nouveau sur l'image. C'est cette méthode 2D (cf algorithme 1) que nous étendrons afin de pouvoir travailler sur des maillages 3D.

## 2.2. Livewire 3D

Une fois que nous avons choisis les sommets sur le maillage qui représentent des parties du contour, nous devons appliquer l'algorithme d'Intelligent Scissoring pour relier ces sommets par des arêtes. Afin de relier tous ces sommets, l'algorithme sera appelé de manière itérative (cf algorithme 3) et le nombre d'appels sera égal au nombre de sommets choisis. Par exemple si nous avons deux sommets, le nombre d'appel sera de deux : un pour relier les deux sommets, et un pour fermer le contour.

À la manière d'un Dijkstra, nous devons considérer toutes les arêtes du maillage pour calculer le meilleur chemin entre 2 sommets. En effet nous donnerons un poids à chaque arête du maillage, et calculerons un chemin de coût minimal. Mais au lieu d'utiliser la distance entre deux sommets pour calculer les coûts comme avec Dijkstra, nous utiliserons une fonction de coût modulable selon des critères (décrise dans la section 2.3) appliqués aux arêtes.

La forme de l'algorithme (cf algorithme 3) est calquée sur la méthode du Livewire 2D (cf algorithme 1), mais au lieu de considérer le N-voisinage de pixels, on considère les sommets adjacents à un sommet (le 1-ring) en utilisant les arêtes qui les relient. C'est d'ailleurs avec ces arêtes que nous allons appliquer des critères de coût pour choisir un chemin. On notera par ailleurs que devant considérer tous les chemins d'arêtes, il est fortement conseillé de nettoyer le maillage au préalable, afin de supprimer les composantes non connexes.

---

### Algorithm 1 calculLiveWire2D

---

**input :**

$p$  : Pixel Seed

$\text{costFunction}(q, r)$  :

Local cost function for link between pixels q and r.

**data structures :**

$L[]$  : List of active pixels sorted by total cost (initially empty).

$N[]$  : Neighborhood set of 8 neighbours of q

$\text{visited}[]$  :

Boolean function indicating if q has been expanded/processed.

$\text{costTo}[]$  :

Total cost function from seed pixel to q.

**output :**

$\text{paths}[]$  : Pointers from each vertex indicating the minimum cost path.

$\text{costTo}[p] = 0$ ;

$L += p$ ;

**while**  $L \neq \emptyset$  **do**

$q = \min(L)$ ;

$\text{visited}[q] = \text{TRUE}$ ;

**for** each  $r \in N[q]$  such that not  $\text{visited}[r]$  **do**

$\text{tmpCost} = \text{costTo}(q) + \text{costFunction}(q, r)$ ;

**if**  $r \in L$  and  $\text{tmpCost} < \text{costTo}[r]$  **then**

$L -= r$

**end if**

**if**  $r \notin L$  **then**

$\text{costTo}[r] = \text{tmpCost}$ ;

$\text{paths}[r] = q$ ;

$L += r$ ;

**end if**

**end for**

**end while**

---



---

### Algorithm 2 draw contour on mesh

---

**Require:**  $\text{contour} // \text{contour of vertices}$

$v \leftarrow \text{startVertex}$

**while**  $v2! = \text{endVertex}$  **do**

$v2 = \text{getNextVertexOf}(v)$

$\text{paths} = \text{calculLivewire3D}(v)$

$\text{drawPath}(\text{paths}, v, v2)$

$v \leftarrow v2$

**end while**

$\text{paths} = \text{calculLivewire3D}(\text{endVertex})$

$\text{drawPath}(\text{paths}, \text{endVertex}, \text{startVertex})$

---

---

**Algorithm 3** calculLiveWire3D

**input :**

- $v$  : Vertex Seed
- $\text{costFunction}(q, r, \text{edge})$  :

Local cost function for link between vertices q and r.

**data structures :**

- $L[]$  : List of active vertices sorted by total cost (initially empty).
- $N[]$  : Neighborhood set of q (contains 1-ring neighbors of the vertex).
- $\text{visited}[]$  :

Boolean function indicating if q has been expanded/processed.

- $\text{costTo}[]$  :

Total cost function from seed vertex to q.

**output :**

- $\text{paths}[]$  : Pointers from each vertex indicating the minimum cost path.

```

 $\text{costTo}[v] = 0;$ 
 $L += v;$ 
while  $L \neq \emptyset$  do
     $q = \min(L);$ 
     $\text{visited}[q] = \text{TRUE};$ 
    for each  $r \in N[q]$  such that not  $\text{visited}[r]$  do
         $\text{edge} = \text{getEdgeBetween}(q, r)$ 
         $\text{tmpCost} = \text{costTo}(q) + \text{costFunction}(q, r, \text{edge});$ 
        if  $r \in L$  and  $\text{tmpCost} < \text{costTo}[r]$  then
             $L -= r;$ 
        end if
        if  $r \notin L$  then
             $\text{costTo}[r] = \text{tmpCost};$ 
             $\text{paths}[r] = q;$ 
             $L += r;$ 
        end if
    end for
end while

```

---

### 2.3. Critères

Notre fonction de coût est représentée par l'équation :

$$\text{cost}(e) = c_{\text{len}}(e) \times c_{\text{ang}}(e) \times c_{\text{curv}}(e) \times c_{\text{angEE}}(e) \times c_{\text{dot}}(e) \times c_{\text{vis}}(e)$$

### 2.4. Longueur d'arête

$$c_{\text{len}}(e)$$

Ce critère permet de favoriser des chemins plus courts, en donnant des faibles coûts aux arêtes les plus courtes.

### 2.5. Angle dièdre

$$c_{\text{ang}}(e) = \frac{\theta_e}{2\pi}$$

Ce critère est pratique lorsqu'on traite des maillages larges/grossiers qui contiennent de fortes informations topologiques comme des zones de plis. Pour des maillages, ou

parties de maillage, plus réguliers on préférera appliquer un critère de courbure.

### 2.6. Courbure

Soit  $v_1, v_2$  les deux sommets de l'arête  $e$ , et  $K_{\text{curv}}(v)$  la courbure gaussienne au sommet  $v$ . Le coût du critère de courbure est représenté par :

$$c_{\text{curv}}(e) = \frac{K_{\text{curv}}(v_1) + K_{\text{curv}}(v_2)}{2}$$

Ce critère, plus adapté pour des maillages plus réguliers, permet de favoriser les chemins en fonction de la courbure gaussienne présente autour de l'arête.

### 2.7. Angle entre arêtes

$$c_{\text{angEE}}(e)$$

Ce critère permet de privilégier les chemins dont les angles entre chaque arête adjacente soit le plus grand possible dans le but d'obtenir un chemin plus court.

### 2.8. Orientation de la normale

$$c_{\text{dot}}(e)$$

Ce critère permet d'encourager le chemin de moindre coût à traverser l'arrière du maillage, c'est-à-dire de choisir un chemin qui passe par la partie non visible par l'utilisateur. De plus, sans ce paramètre, le chemin choisi pourrait avoir tendance à seulement suivre les bords visibles du maillage. Le principe repose sur le fait de donner un moindre coût aux arêtes, dont les normales de leurs faces adjacentes sont au mieux alignées avec la direction du point de vue de l'utilisateur. Ce critère sera surtout utilisé pour fermer un contour et encourager le chemin à ne pas repasser sur la partie du contour déjà tracée.

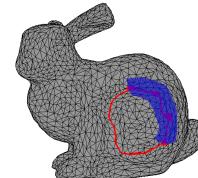


Figure 1: Application du Livewire avec notre méthode. Le Livewire est appliqué selon un contour défini autour de la cuisse du lapin. En bleu on observe la zone de visibilité.

### 2.9. Visibilité

$$c_{\text{vis}}(e)$$

Ce critère donne un moindre coût aux arêtes non-visibles. Avec  $c_{\text{dot}}(e)$ , le critère de visibilité encourage le contour de moindre coût à traverser la partie arrière (non-visible) du maillage.

Initialement dans la thèse de W. Kiefer [Kie04], la visibilité est définie par une zone d'épaisseur constante couvrant sur tout son long le tracé effectué par l'utilisateur. On peut

comparer cela au tracé d'un pinceau dans des logiciels de dessin. Dans de tels logiciels, l'épaisseur de la brosse du pinceau est représentée par un rayon. Pour *W. Kiefer*, l'épaisseur de ce tracé indique une zone dite de visibilité. En dehors de cette zone, les arêtes du maillage traitées sont considérées comme non-visibles. Par conséquent, dans notre projet, nous modélisons ce critère en fonction de la localisation de l'arête.

Dans notre projet contrairement aux travaux originaux de la publication [Kie04] nous n'avons pas besoin d'un tracé de l'utilisateur. Nous définissons l'épaisseur du trait par un seuil de distance. Ce seuil représenterait alors le rayon d'épaisseur du pinceau. La ligne centrale de ce tracé est modélisée par une succession de points, donnés en entrée de l'application. Cette succession de points forme un chemin. Ce chemin est trouvé grâce à l'algorithme de *Dijkstra* [Dij59]. Le seuil marque la zone de visibilité tout le long de ce chemin. Dans notre programme cette zone est virtuelle et invisible (représentée en bleu dans la Figure 1).

Les arêtes localisées à l'extérieur de cette zone de visibilité, ont un coût qui tend vers 0 d'autant plus que leur distance à la ligne centrale du tracé est grande. En revanche, une arête située dans la zone de visibilité aura un coût tendant moins rapidement vers 0 que les arêtes non-visibles. Ainsi une arête dite visible aura un coût relativement plus proche de 1.

## 2.10. Distance à la ligne centrale du tracé

$$c_{dist}(e) = \frac{r - d}{r}, r \text{ le rayon du tracé, } d \text{ la distance à la ligne centrale du contour/tracé}$$

La ligne centrale du contour est le chemin passant par une succession de points donnés en entrée. Ce chemin est trouvé à l'aide de l'algorithme de *Dijkstra* [Dij59]. Pour une arête donnée  $e$ , le critère de distance à la ligne centrale du tracé représente la volonté de négliger les arêtes proches de la ligne centrale.

## 3. Identifier la partie à segmenter

### 3.1. Identification

Après avoir décrit le fonctionnement de l'algorithme du livewire 3D, il est intéressant de pouvoir choisir avec précision les composantes que l'on veut segmenter. Comme dit dans les sections précédentes, nous ne requérons pas d'interactions avec un utilisateur. En effet, le but serait plutôt d'appliquer l'algorithme de façon automatique, à l'aide de contours pré-enregistrés.

Il se trouve qu'il existe déjà un algorithme utilisant de l'intelligence artificielle, qui permet d'isoler des parties d'un nuage de points [MLT19]. L'inconvénient de cette méthode est qu'elle ne peut pas s'appliquer sur un maillage. Mais avec quelques étapes supplémentaires, nous pensons que nous pouvons retrouver la composante issue du nuage de point, sur un maillage créé à partir du même nuage.

Nous précisons que les composantes des nuages de points



Figure 2: Nuage de points de Siva, avec détection de parties morphologiques.

étudiés, nous ont été fournies au préalable. En effet notre travail consiste surtout à transposer ce résultat sur un maillage. Pour cela, une première étape sera d'extraire les points composants la bordure du nuage segmenté, et de les projeter sur des sommets du maillage à des positions équivalentes. Une seconde étape sera de relier logiquement par des arêtes les sommets trouvés sur le maillage.

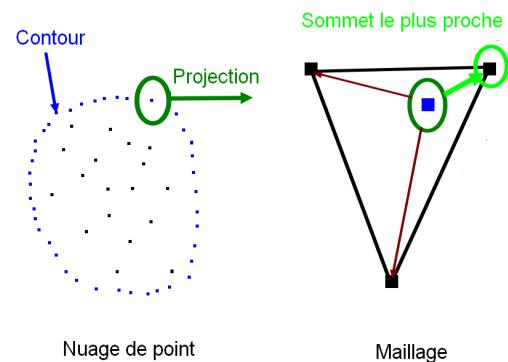


Figure 3: Projection de points d'intérêts 3D en bleu, du nuage de point vers le maillage. Une fois projeté, le point est attribué à un sommet du maillage en fonction d'un test de distance.

### 3.2. Extraction de contour

Avant de pouvoir projeter, il faut déterminer les points d'intérêts 3D qui constituent le contour du nuage. Cela peut se faire manuellement à l'aide d'un visualisateur, grâce à des outils comme Meshlab ou Blender, mais nous favoriserons une méthode plus automatique.

Nous avons opté pour une extraction de bordure utilisant du Range Imaging [PCL]. Le Range Imaging est en fait le

nom d'une collection de techniques utilisées pour produire une image 2D à partir d'une scène 3D. La méthode que l'on utilise permet d'extraire les points représentant la bordure du nuage.

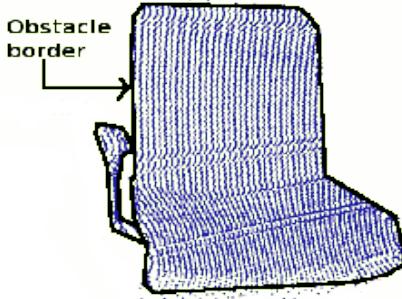


Figure 4: Schéma de Range Imaging

### 3.3. Projection

Une fois que nous avons extrait les points d'intérêt 3D, il s'agit de pouvoir les projeter sur le maillage. C'est à dire d'extraire les sommets et les arêtes qui satisfont géométriquement la sélection du contour. Sachant que le maillage a son origine calquée sur celle du nuage de point, ceci va nous faciliter la tâche. Nous utiliserons donc un algorithme de distance min (décris ci-dessous) entre les points d'intérêts et tous les sommets du maillage, afin d'extraire les sommets qui correspondent au mieux à la position de ces points.

---

#### Algorithm 4 Projeter les points d'intérêt 3D sur le maillage

```

for every point  $p$  of the point cloud do
    for every vertex  $v$  of the mesh do
         $dist \leftarrow calculateDistanceBetween(p, v)$ 
        if  $dist < minDist$  then
             $minDist \leftarrow dist$ 
             $minV \leftarrow v$ 
        end if
    end for
     $contour+ = minV$ 
end for

```

---

Selon la taille et la topologie du maillage, plusieurs points peuvent être attribués au même sommet. Il est vivement conseillé de ne pas enregistrer de doublons, afin de ne pas perturber l'algorithme du Livewire. On précise par ailleurs que l'on aura pas forcément le même nombre de sommets que de points d'intérêts.

Selon comment le contour a été construit, les sommets ne seront probablement pas tous ajoutés dans l'ordre. Afin de remédier à cela, nous avons choisi d'opter pour une solution à base d'un algorithme (décris ci-dessous) qui les remplacera dans l'ordre. Cette méthode permet d'obtenir des résultats simples mais pas forcément optimisés.

Une fois que nous avons tous ces sommets, nous devons pouvoir retracer un contour contenant les arêtes et les faces qui composeront la partie à segmenter sur le maillage.

---

#### Algorithm 5 Réordonner les sommets du contour

```

contour2  $\leftarrow$  contour
clearContour(contour)
 $v \leftarrow getFirstVertexOf(contour2)$ 
contour2  $\leftarrow= v$ 
contour+ = v
while contour2 is not empty do
     $v2 \leftarrow searchVertexMinDistFrom(contour2, v)$ 
     $v \leftarrow v2$ 
    contour2  $\leftarrow= v$ 
    contour+ = v
end while

```

---

C'est à ce moment là que nous appliquer le livewire 3D. Il se trouve qu'il existe déjà un algorithme utilisant de l'intelligence artificielle, qui permet d'isoler des parties d'un nuage de points [MLT19]. L'inconvénient de cette méthode est qu'elle ne peut pas s'appliquer sur un maillage. Mais avec quelques étapes supplémentaires, nous pensons que nous pouvons retrouver la composante issue du nuage de point, sur un maillage créé à partir du m

### 4. Résultats et validation

Afin de réaliser nos tests, nous avons écrit un programme en C++. Les résultats présentés sont des sorties de ce programme, et le tout a été obtenu avec l'utilisation d'OpenGL et de openMesh.

#### 4.1. Résultats et validation du Livewire

Dans la méthode d'origine des ciseaux intelligents pour la segmentation interactive de maillages 3D [Kiefer 2004], l'utilisateur doit dessiner une esquisse du contour - sinon le contour entier. Ensuite, la méthode consiste à prendre cette esquisse de contour en paramètre d'entrée du Livewire et à chercher le chemin de moindre coût.

Il illustre les deux phases principales de son algorithme.

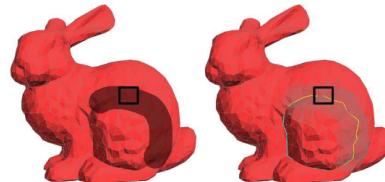


Figure 5: Application du Livewire sur le Stanford Bunny avec la méthode de W. Kiefer [Intelligent Scissoring for Interactive Segmentation of 3D Meshes, Kiefer - 2004, p. 7]

À partir de cela, nous prenons ces images en références pour tester notre implémentation revisitée de l'algorithme. Dans rappelons que dans notre méthode, l'esquisse de l'utilisateur est directement fournie. Nous testons donc notre version du Livewire sur un maillage test de référence, le Stanford Bunny.

Il est nécessaire d'admettre qu'en premier lieu, nos points de contour initiaux, formant la ligne centrale du contour du

tracé, ne reproduisent pas exactement le même tracé que celui qu'aurait pu faire l'utilisateur. En effet, notre test se base essentiellement sur les résultats visuels des tests originaux [Kiefer 2004] car nous n'avons pas accès à plus de données. Remarquons au moins deux biais évidents. Le premier est donc la représentation du tracé dessiné par l'utilisateur. Les sommets sélectionnés sont situés plus à l'extérieur de la cuisse dans les tests originaux. De plus nous ne prenons que quatre points. Augmenter le nombre de points peut augmenter la précision mais permet une observation amoindrie du Livewire dans certaines zones. Le second est le modèle géométrique sur lequel sont effectués les tests. Notre maillage comporte bien moins d'éléments que le maillage test référence. On le remarque au niveau du détail visuel. Pour effectuer des tests intéressants et valider au mieux notre méthode, il faudrait pouvoir mesurer la qualité de notre maillage test et du maillage test référence, afin de travailler sur des modèles quasiment similaires. Ainsi au niveau de la cuisse il faut noter que notre maillage test a un relief bien moins prononcé que le maillage test référence.

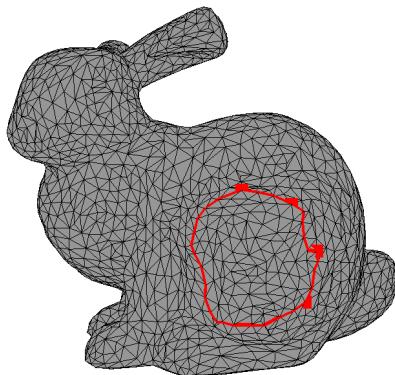


Figure 6: Application du Livewire sur la cuisse du lapin avec notre méthode

Toutefois, il est évident de constater (Figure 6) que l'allure du chemin de moindre coût paraît correcte et englobe bien la partie saillante de la cuisse du lapin. Notre implémentation du Livewire est correcte dans un cas simple comme celui-ci. Il convient d'observer d'autres cas.

Dans le cas illustré sur Figure 7, le Livewire s'effectue correctement autour du cou du lapin. La zone de visibilité est toujours la même. Les points représentant la ligne centrale du contour sont espacés sur trois quarts de la circonférence du cou.

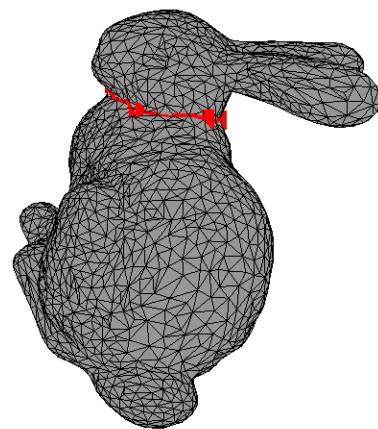


Figure 7: Application du Livewire avec notre méthode. Livewire appliquée autour du cou du lapin

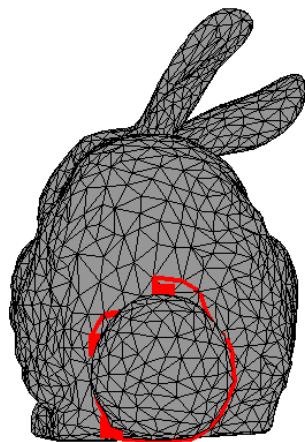


Figure 8: Application du Livewire avec notre méthode. Livewire appliquée autour de la queue du lapin

#### 4.2. Tests Projections et segmentations

Maintenant que nous avons validé notre test du livewire, nous allons pouvoir le tester sur des maillages plus complexes, avec des contours issus de nos méthodes de projection. Les tests suivants se feront sur les maillages représentant les statues "myson" (appelé tel quel en référence de sa provenance)(Fig.9) et siva(Fig.10). Ces deux maillages ont été obtenues à partir de nuages de points, eux-mêmes issus d'un scan avec une méthode de photogrammétrie.



Figure 9: Présentation de la statue de Myson, en photo, nuage de points et maillage



Figure 10: Présentation de la statue de Siva, en photo, nuage de points et maillage

**4.2.0.1. Extraction de contour** Avant de parler des tests avec le Livewire, il peut être intéressant de parler de la qualité des contours obtenus avec nos méthodes de projection.

Les résultats obtenus pour les extractions de contour sur Myson ne sont pas optimaux. Dès lors que l'on a des nuages représentant plusieurs parties trop espacées, on obtient des contours qui ne prennent pas forcément en compte tout le nuage (Fig.11 mask 1). On peut aussi avoir le cas où l'algorithme compte un seul contour au lieu de deux (Fig.11 mask2). Afin de palier à ce problème, on pourra toujours modifier manuellement les contours pour obtenir des résultats plus satisfaisants (Fig.12). Ce sont d'ailleurs ces contours là que l'on retiendra pour tester myson.

Pour les contours des parties de siva, on a globalement des résultats plus précis, notamment pour le n°1 (Fig.13), et le n°6(Fig.13) reconnaît même deux contours. Certains résultats restent quand même décevants comme le n°0(Fig.13).

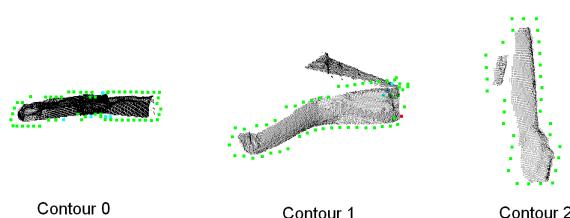


Figure 11: Détection des contours sur des parties originelles de myson. Le contour 1 ne détecte qu'une partie de la forme et le contour 2 ne crée qu'un seul contour au lieu de deux.

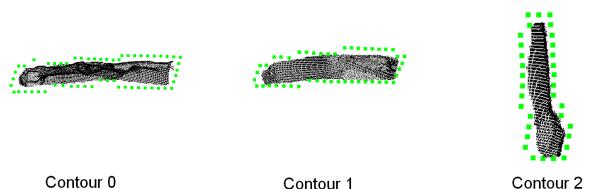


Figure 12: Détection des contours sur des parties modifiées de myson

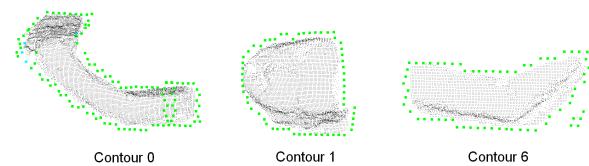


Figure 13: Détection des contours sur des parties de siva. Le contour 0 évalue mal le contour, mais le contour 6 parvient créer deux contours séparés.

**4.2.0.2. Livewire appliqué à myson et siva** Pour toutes les images suivantes, les points verts sont les sommets qui ont été obtenus par projection sur le maillage, les arêtes rouges composent le contour, et les points bleus représentent le contour original en nuage de point (séparé du maillage). Par ailleurs, et pour un souci de visualisation, les points bleus seront un peu décalés par rapport à leur position d'origine.

Nous avons opéré des tests sur des contours de myson représentant ses bras et ses jambes. Les résultats sont plutôt variés. En général, et lorsque nous avons beaucoup de sommets verts, on redélimite assez bien les lignes du contour avec les arêtes rouges. On peut le vérifier sur la figure 14. On notera que, comme notre méthode pour ordonner les sommets n'est pas parfaite, nous aurons parfois quelques soucis comme sur le contour 0, où le livewire essaie de relier des sommets qui ne sont pas du tout proches. Le même soucis apparaît sur le contour 2, mais de manière plus discrète (sommet vert situé au plus haut à droite). Quant à la qualité du tracé par l'algorithme, on voit que celui-ci ne concorde pas parfaitement avec le nuage de point. Cela n'empêche pas les résultats d'être intéressants et l'on remarque, comme sur le contour 1, que certains chemins d'arêtes ne suivent pas seulement les chemins les plus courts, mais aussi les courbures.

Pour les tests sur Siva, nous avions à tester des parties comme ses bras ou son visage. La particularité de ce maillage, comparé à celui de My Son, est qu'il comporte plusieurs trous. Au final, cela pose peu de problèmes au Livewire. On peut alors voir que certaines parties se tracent assez bien, comme pour le contour 1 (Fig.16) et le contour 5 (Fig.17). On peut même voir que le contour 5 arrive à recomposer le contour malgré un très gros trou proche de son contour. En revanche pour le contour 0 (Fig.15) les résultats sont plus discutables, notamment en raison d'un contour de points bleus mal calculé. Malgré cela, il est tout de même

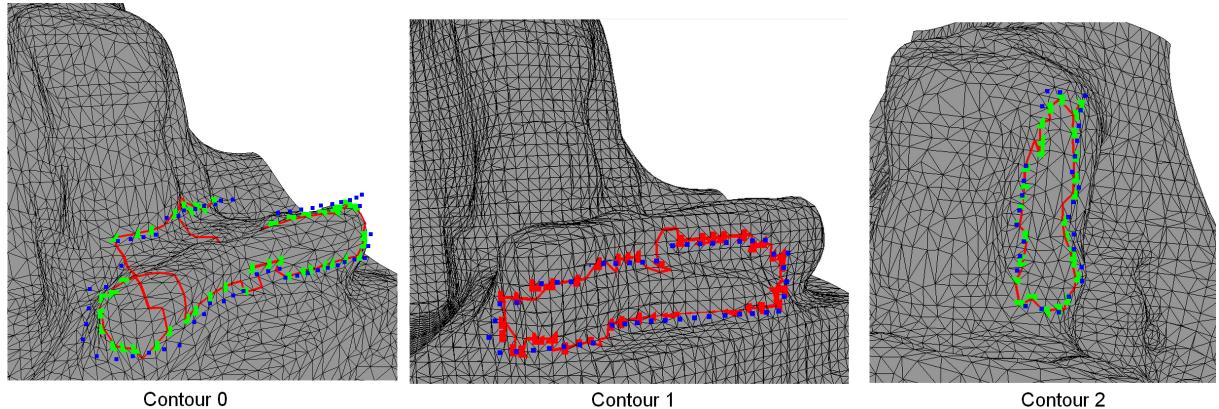


Figure 14: Livewire sur les contours 0, 1 et 2 de My Son. Le contour 0 semble avoir quelques soucis à cause d'un contour de sommets mal ordonnés. Les contours 1 et 2 semblent donner de meilleurs résultats.

intéressant de voir que le tracé d'arêtes rouge garde un minimum de logique, et que même s'il est déformé, le tracé garde une forme rappelant le bras de la statue.

Nous avons aussi pu tester ce qu'il se passait si l'on réduisait drastiquement le nombre de points d'intérêt 3D projetés en tant que points verts (Fig.18). Dans certains cas, comme pour le contour 5 de Siva, les résultats restent assez cohérents. Mais ce n'est pas le cas pour d'autres, comme pour le contour 1 de My Son. Bien sûr, il reste quelques problèmes comme le contour de sommets verts qui n'est pas toujours dans l'ordre, mais en général les résultats sont largement moins bons. Nous pouvons tout de même remarquer que, même si parfois le tracé d'arêtes rouges ne correspond plus au contour de points bleus, le Livewire crée des chemins qui respectent les critères que nous avons pu lui donner. En effet, on peut voir sur le contour 1 de My Son (contour My Son 1 moyen sur la figure 18) que la forme tracée reste cohérente lorsque l'on a peu de sommets, mais préfère choisir certains chemins où la courbure est plus forte. Cela est encore plus flagrant lorsqu'on baisse encore le nombre de sommets verts (contour My Son 1 faible sur la figure 18).

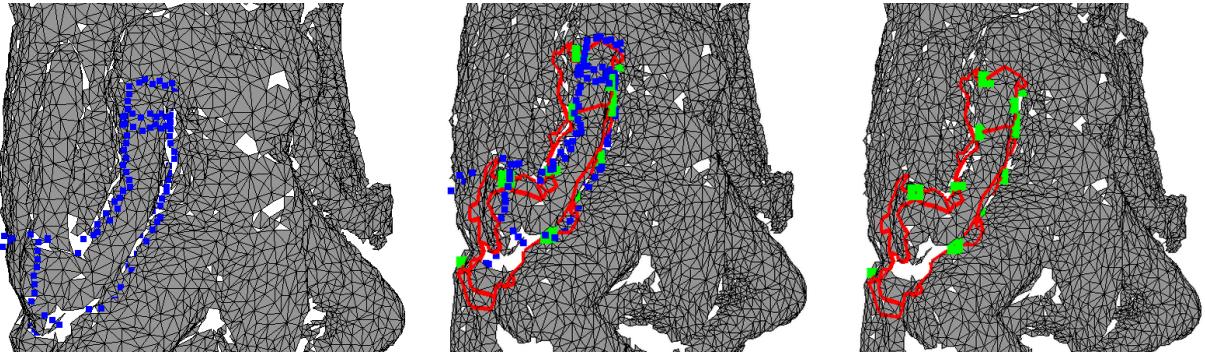


Figure 15: Livewire sur le contours 0 de Siva. Ici, le tracé du contour ne parvient pas à donner un résultat assez satisfaisant, la faute à un contour de points bleus qui n'a pas bien été calculé.

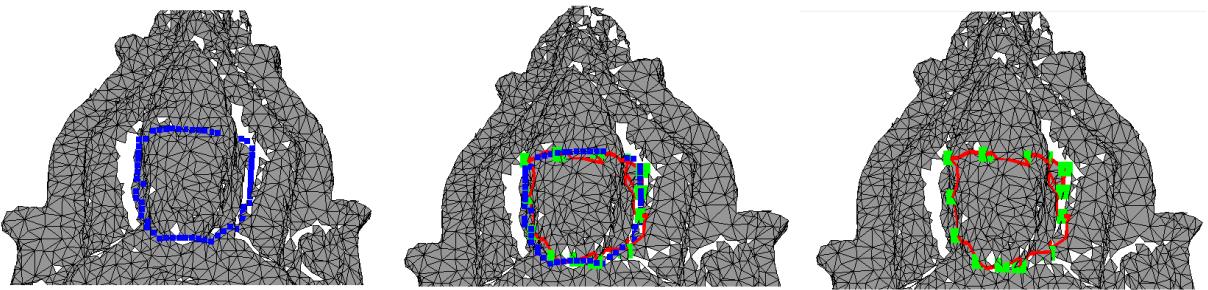


Figure 16: Livewire sur le contours 1 de siva. Le tracé de contour semble bien fonctionner même s'il ne semble pas épouser totalement la forme du contour de points bleus.

#### 4.2.1. Limites de notre méthode

Notre méthode connaît aussi des limites. Ces limites sont liées aux choix de notre implémentation de l'algorithme du Livewire et aux contraintes liées aux données de travail dont nous disposons.

**4.2.1.1. Limites liées à la mise en oeuvre du Livewire**  
 Dans la Figure 19, on observe qu'en absence de contraintes fortes imposées par l'esquisse du contour (donnée en entrée du programme), les critères définissent plus fortement le choix du chemin de moindre coût. Dans la méthode originale du Livewire 3D [Kie04] il n'y a que très peu de détail sur la modélisation mathématique et algorithmique des critères. Ainsi dans nos travaux, nous avons simplement suivi les indications dont nous disposions pour définir et mettre en oeuvre ces critères. Nos calculs de critères sont donc fondés sur nos connaissances acquises et sur notre expérience. La figure mentionnée plus tôt illustre une faiblesse dans notre implémentation des critères, notamment en raison de l'absence de détails concernant la manière dont ils devraient être modélisés. Chacun de ces critères servent à définir le coût d'une arête.

La mise en oeuvre de notre programme est également limitée par la mémoire disponible sur la machine qui l'exécute. Nous préchargeons en mémoire les données qui seront sollicitées plusieurs fois pour des calculs : ce sont des données comme les angles entre arêtes ou les normales aux sommets par exemple. Évidemment, plus le maillage dense, plus il y aura de données à précharger. La topologie du maillage en entrée doit être respectée, c'est pourquoi nous ne faisons pas d'opération de remaillage. Il convient de préciser

que nous travaillons sur un maillage irrégulier du *Stanford Bunny* de basse résolution. Cela nous permet de travailler plus efficacement au vu des performances de nos machines. Toutefois une basse résolution et une irrégularité dans les propriétés du maillage peuvent également affecter nos résultats dans les cas où les critères du Livewire sont fortement sollicités.

Notre implémentation du Livewire est aussi limitée par le contour simulant le tracé de l'utilisateur, fournit en entrée.

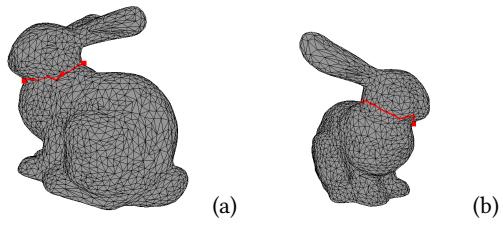


Figure 19: (a) La partie du chemin de moindre coût passant par les points du contour passe correctement sur le cou du lapin.

(b) La partie du chemin de moindre coût soumis à moins de contraintes emprunte un chemin sur la tête du lapin.

**4.2.1.2. Limites liées aux données** L'évaluation de la performance de notre Livewire dépend aussi des données fournies en entrées. Notamment lorsque la qualité du maillage est dégradée. La qualité est dégradée lorsque la surface est discontinue. Le maillage de la statue de Siva

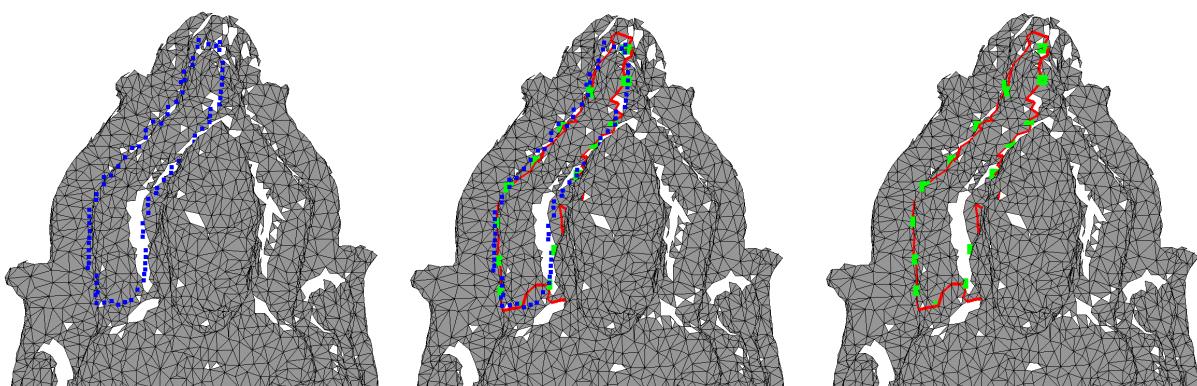


Figure 17: Livewire sur le contours 5 de siva. Le tracé du contour arrive à rester plutôt cohérent malgré un gros trou dans le maillage.

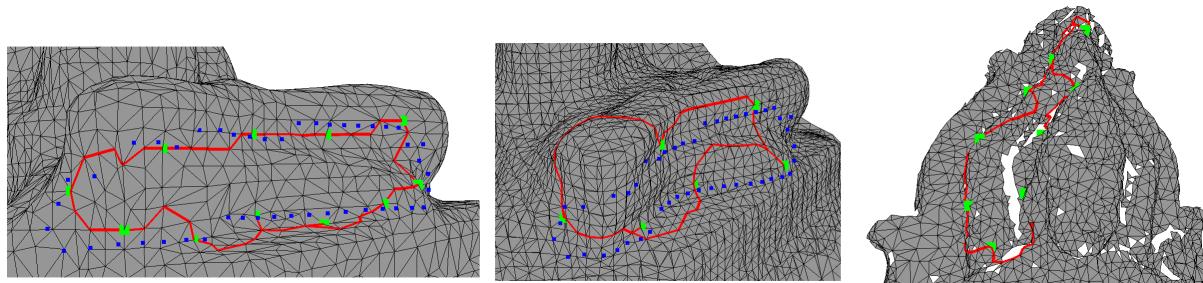
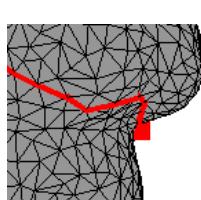


Figure 18: Livewire sur des contours réduits. Les résultats sont plutôt variés dès lors qu'on baisse la quantité de sommets verts.



produit également dans les cas où le maillage a plusieurs composantes connexes.

Figure 20: Application du Livewire avec notre méthode. Le Livewire est appliqué autour du cou du lapin avec un tracé de contour moins précis. Le chemin de moindre coût est soumis à moins de contraintes et passe sur la tête du lapin.

est généré grâce à la reconstruction d'une surface à partir d'un nuage de point. Toutefois le maillage résultant de cette reconstruction présente des artefacts, ou effets indésirables. Remarquons deux phénomènes problématiques qui surviennent pour Siva (Figure 21 et 22). Premier phénomène, lorsque nous projetons les points du contour fournis en entrée (appartenant au nuage de point avant reconstruction) sur le maillage, les points projetés ne sont pas toujours bien retrouvés. En effet ce problème est engendré par la discontinuité de la surface reconstruite. Le point du contour est projeté vers le sommet le plus proche. Or s'il y a un trou (discontinuité de la surface), le point du contour est projeté sur un sommet indésirable dans le pire des cas. C'est le second phénomène problématique. Un tel événement déstabilise le choix du chemin de moindre coût et le chemin adopte un itinéraire incohérent. Ce comportement du programme se

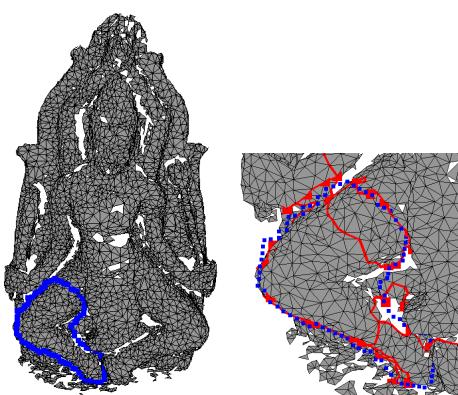


Figure 21: Application du Livewire avec notre méthode. Le Livewire est appliqué selon un contour défini autour de la jambe droite de la sculpture de Siva. La qualité du maillage influence visiblement le chemin de moindre coût.

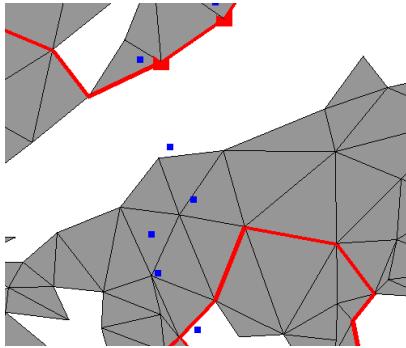


Figure 22: Application du Livewire avec notre méthode. Le Livewire est appliqué selon un contour défini autour de la jambe droite de la sculpture de Siva. Le nuage de point en bleu indique le contour donné en entrée du programme. En rouge, le chemin de moindre coût.

## 5. Améliorations

### 5.1. Améliorations mineures

Plusieurs améliorations mineures sont possibles pour notre projet.

Des améliorations simples comme un travail sur la qualité des critères peut être effectué. Une telle amélioration consisterait à trouver un algorithme efficace pour chaque critère. Pour le critère de visibilité il faut trouver à la fois un bon seuil de visibilité autour de la ligne centrale du contour, mais aussi un calcul robuste qui varie correctement en fonction de si l'arête est visible ou non. Il convient aussi de déterminer des bornes pour les critères de coût. Une réflexion nous mène aussi à nous demander si la valeur d'un critère doit être compris entre 0 et 1, ou entre 0 et  $2\pi$  par exemple. Il faut donc trouver pour chaque critère un intervalle de valeurs qui permet d'influencer faiblement ou fortement le coût total final. C'est aussi un moyen de fournir une activité pédagogique à des étudiants sous forme de simple projet destiné à alimenter la curiosité informatique.

### 5.2. Sélection des points de contour au clic

Cette amélioration serait idéale pour avoir une application orientée expérience utilisateur.

Notre consigne était de limiter au maximum les interactions avec l'utilisateur. À défaut de ne pas pouvoir reproduire l'esquisse d'un contour de manière fluide, notre projet peut être amélioré en offrant la possibilité à l'utilisateur de choisir une suite de point sur le maillage. Pour cela il suffirait de connaître la position de la caméra de l'utilisateur, pour chaque clic de souris sur un pixel de l'écran. On lance ensuite un rayon (principe du *ray-casting*) du point de vue utilisateur vers la projection 3D dans la scène du pixel sélectionné. Cela donne un vecteur, si le rayon projeté dans la direction de ce vecteur entre un collision avec un point du maillage, alors on cherche le sommet le plus proche du point de collision.

Nous avons implémenté une version *Qt - OpenMesh* de l'algorithme d'intersection « rayon-triangle » de Möller-Trumbore [MT05]. Elle est disponible dans notre projet. Elle n'a toutefois pas été testée. C'est une fonctionnalité dormante mais qui offre des perspectives d'utilisations.

Cet algorithme permet de trouver rapidement l'intersection d'un rayon sur un triangle d'un maillage sans nécessiter le pré-calcul de l'équation du plan contenant le triangle. Ce qui est idéal pour les maillages qui contiennent beaucoup de triangles.

### 5.3. Suppression de sous-maillages non-connexes au maillage principal

Une autre amélioration intéressante, pour les maillages reconstruits, est de pouvoir enlever les composantes connexes résiduelles liées à la reconstruction du maillage à partir du nuage de point. Une telle amélioration permettrait au Livewire de ne pas avoir à chercher un chemin inexistant, et au contour de se projeter sur un sommet valide.

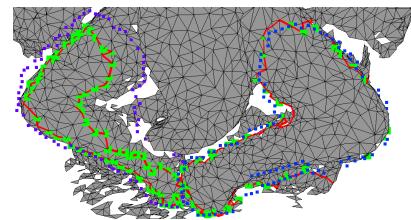


Figure 23: Application du Multi-Livewire avec notre méthode. On effectue deux livewires de manière séquentielle en chargeant les contours 2 et 3 de Siva.

### 5.4. Multi-livewire

Nous avons pu faire une version expérimentale et évoluée de notre algorithme. Cette version permet d'effectuer plusieurs livewires, de manière séquentielle sur le maillage et à partir de plusieurs contours. Toutefois le Multi-livewire permettrait de pouvoir segmenter plusieurs partie d'un maillage si les contours sont déjà fournis. Toutefois différentes problématiques surviennent pour une telle méthode.

## 6. Conclusion

Afin de pouvoir projeter de façon propre sur des maillages, des contours issus de nuages de points, nous avons développé notre méthode de livewire 3D. Le but était de pouvoir recomposer un contour sur un maillage, de façon à respecter la surface et topologie de ce dernier. Nous avons d'abord procédé à des tests sur des maillages simples pour valider le développement de notre algorithme, et les résultats ont été probants. Pour l'utilisation de contours complexes, nous avons utilisé une méthode de PCL [PCL] afin d'extraire les points d'intérêts du nuage de point, et de pouvoir les appuyer sur le maillage, à des sommets qui délimitent le contour. Nous avons alors obtenu des résultats assez variés, dont certains décevants. Cela pouvait s'expliquer en général par des reconnaissances de contour trop simplifiés, ou à des réglages laborieux de l'algorithme. Mais au final, certains résultats démontrent une certaine efficacité de la projection, ainsi que de la reconstruction d'un contour par le livewire 3D. Ainsi, malgré quelques difficultés, nous pensons que notre méthode peut très bien s'appliquer au problème posé, et qu'elle pourrait encore être améliorée de bien des façons.

---

## 7. Travaux Futurs

### 7.1. Traitement *Out-of-core* de maillage haute résolution pour le Livewire

On remarque qu'en dehors des capacités de calcul, les maillages haute résolution, destinés plutôt à la recherche, ou bien à l'industrie, occupent beaucoup d'espace en mémoire principale. Parfois ils la saturent. Lorsque les données ne peuvent plus être contenues en mémoire, on peut traiter notre maillage avec des méthodes dites *Out-of-core* ou bien *hors-mémoire*. Une méthode intéressante mais qui demanderait une charge de travail en amont du Livewire, serait le paradigme des séquences de traitement [IGS20] pour les maillages de haute résolution. Ce paradigme consiste à réordonner l'arrangement des triangles pour pouvoir traiter plus rapidement le maillage. Dans ce cas de figure, la mémoire principale est utilisée comme un cache contenant la poignée de données à traiter.

### 7.2. Gestion des recouvrements de zones pour le Multi-Livewire

Pour des segmentations plus spécifiques avec le Multi-livewire, il peut arriver que plusieurs chemin de moindre coût entrent en conflit et se disputent une arrête ou un sommet ou bien une zone plus large. Il y a différents cas de figures possibles pour régler un tel problème. Si la zone d'intersection des deux circuits  $c_1$  et  $c_2$  (chemins de moindre coût) contient une face ou plus alors il faut soit rétrograder la surface de  $c_1$  située dans  $c_2$  au dernier sommet de  $c_1$  avant de pénétrer dans la surface de  $c_2$ , le contraire est aussi envisageable. Ou bien il est aussi possible de fusionner les deux circuits en un en fonction des propriétés du Livewire. Dans l'idée, cela reviendrait à reproduire le système des recouvrements de contours sur les logiciels de dessin et retouche d'image, avec différentes options possibles pour les interactions entre contours comme l'union ou la différence. Une version de l'algorithme de Weiler-Atherton [WA77] pour les maillages, est une bonne issue et s'inscrit dans le principe de la transformation d'outils 2D pour la 3D. Toutefois le fait de devoir traiter les conflits entre différents circuits (par extension des *polygones*) sur la surface du maillage, pourrait empêcher une parallélisation efficace de plusieurs livewires en simultané pour le Multi-Livewire.

### 7.3. Projection des points d'intérêt

Projection des points d'intérêt selon la normale aux faces. Si on subdivise les triangles correspondants aux faces retrouvées, on obtient une meilleure approximation des résultats pour la méthode de projection. Par ailleurs dans la thèse de la segmentation 3D par Livewire, [Kie04] l'auteur indique qu'il effectue un remaillage (*flip*, *swap*, *collapse*) dans toute la zone dite visible. C'est à dire, un remaillage sur les éléments du maillage dans l'épaisseur du tracé de l'utilisateur. Ce remaillage pourrait être plus efficace qu'une simple subdivision. Les deux méthodes sont à évaluer afin d'affiner la précision du Livewire.

Se rattacher aux sommets du maillage le plus proche du point d'intérêt dépend de la qualité du maillage et induit de l'erreur comme nous l'avons montré précédemment.

## Références

- [Bag07] BAGGIO D. L. : *GPGPU Based Image Segmentation Livewire Algorithm Implementation*. PhD thesis, Campo Montenegro, São José dos Campos, SP - Brazil, 2007.
- [Dij59] DIJKSTRA E. W. : A note on two problems in connexion with graphs. *Numerische mathematik*. Vol. 1, Num. 1 (1959), 269–271.
- [FM06] FILIP MALMBERG ERIK VIDHOLM I. N. : *A 3D Live-Wire Segmentation Method for Volume Images Using Haptic Interaction*. PhD thesis, 2006. <http://www.cb.uu.se/filip/Thesis/PaperI.pdf>.
- [Gra06] GRADY L. : Minimal surfaces extend shortest path segmentation methods to 3d. 9. <http://leogrady.net/wp-content/uploads/2017/01/grady2006minimal.pdf>.
- [IGS20] ISENBURG M., GUMHOLD S., SNOEYINK J. : Processing sequences : A new paradigm for out-of-core processing on large meshes.
- [Kie04] KIEFER W. : *Intelligent Scissoring for Interactive Segmentation of 3D Meshes*. PhD thesis, Princeton University, April 2004.
- [MLT19] MAU LE TIEN KHOI NGUYEN TAN R. R. : *Matching correspondence between images and 3D model in a reconstruction process*. PhD thesis, University of Danang, April 2019. pp.64-69. 0.31130/jst.2016.29 Hal-02113975 <https://hal.archives-ouvertes.fr/hal-02113975/document>.
- [MT05] MÖLLER T., TRUMBORE B. : Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*. Vol. 2 (08 2005).
- [PCL] PCL : How to extract borders from range images. <http://pointclouds.org/documentation>.
- [WA77] WEILER K., ATHERTON P. : Hidden surface removal using polygon area sorting. *SIGGRAPH Comput. Graph..* Vol. 11, Num. 2 (juillet 1977), 214–222.
- [Wik] WIKIPÉDIA : Livewire segmentation technique. [https://en.wikipedia.org/Livewire\\_Segmentation\\_Technique](https://en.wikipedia.org/Livewire_Segmentation_Technique).