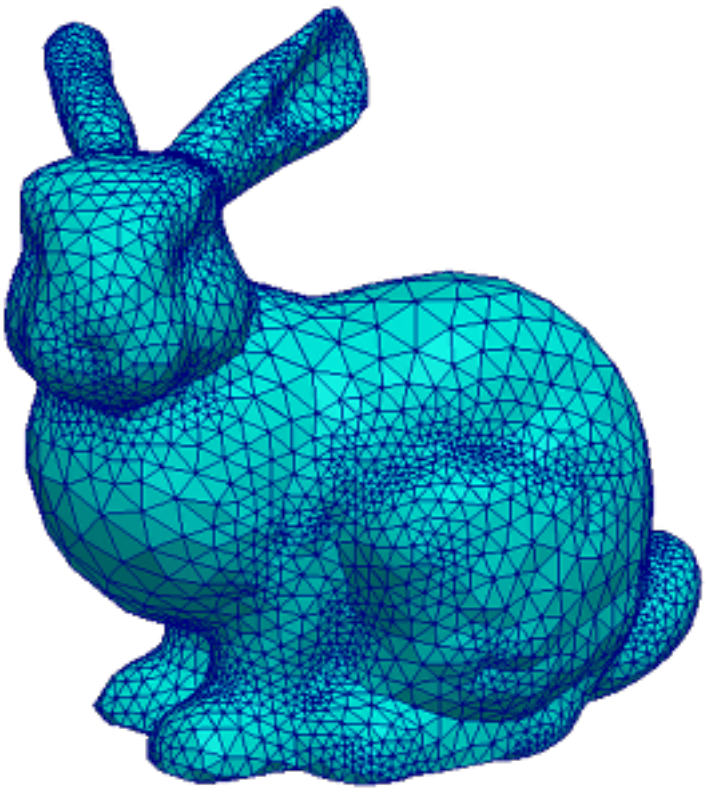
Rapport de TER

Propriétés de formes

Antoine Lemoine Olivier Raizer
Tuteur : Jean-Paul Caltagirone

2009 - 2010



Propriétés de formes

Antoine Lemoine Olivier Raizer
Tuteur : Jean-Paul Caltagirone

En mécanique des fluides numérique, les méthodes lagrangiennes de représentation d'interfaces requièrent l'utilisation de maillages complexes. Ce rapport de TER traite de techniques permettant la manipulation de maillages surfaciques 3D. Dans une première partie, nous explorons les capacités de la bibliothèque CGAL [1], écrite en C++, à traiter de ce sujet. Et dans une seconde partie, nous présentons des algorithmes comme le raffinement de maillages, le lissage à volume constant et la courbure. Les résultats des tests sur des maillages célèbres sont présentés dans ce rapport.

In CFD, lagrangian methods of interface representation require the use of complexe meshes. This report deals with some technics manipulating 3D surface meshes. In the first part of our report, we explore the abilities of the CGAL library [1], written in C++, to treat this topic. In the second part of our report, we explain some algorithms *e.g.* mesh refinement, isovolume smoothing and curvature computing. Our results will be presented on famous meshes.

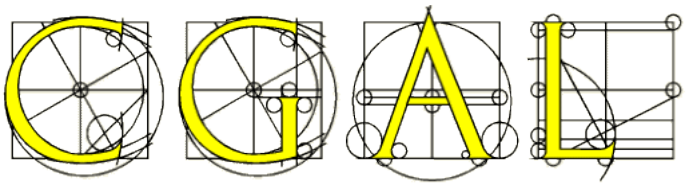
Table des matières

1 Abstract

1.1 Introduction

In some applications of CFD, like lagrangian representation of interface, the use of complex meshes is needed and different operations must be applied on them. Many tools are presented in this report, *e.g.* curvature computing, isovolume smoothing and mesh refinement. This report is composed of two main parts. The first one is the presentation of *CGAL* library [1] and the other one is the isovolume smoothing.

1.2 CGAL library



CGAL (Computational Geometry Algorithms Library) is a geometric library written in C++ by INRIA. This library offers data structures and algorithms like triangulations, Voronoi diagrams, mesh generation, mesh refinement, shape analysis, fitting, and distances. More details are available on [1]. In our work, we used the 3.5.1 version of this library on Ubuntu (9.04, 9.10 and 10.04).
Our goal in the exploration of this library was to find algorithms like isovolume smoothing.

Our main results with this library are the refinement and CSG (Constructive Solid Geometry) operations.
None of refinement method implemented in this library allows both volume conservation and surface quality improving. However, one of these algorithms catches our attention. This is the 1-to-4-subdivision (see figure (1)).

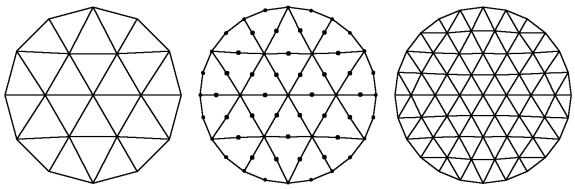


FIGURE 1 – 1-to-4-subdivision algorithm. (a) Initial conditions. (b) Computation of edge centers. (c) All new centers are linked together in each face.

This algorithm has the ability to be volume conservative.

For CSG operations, an example of code can be shown in the listing (1). It substracts a mesh to an other. The figure 2 give an example.

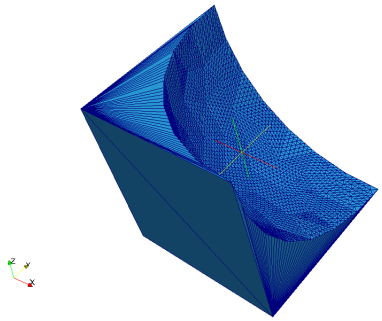


FIGURE 2 – Substraction of a cube by a sphere by using the program (1).

1.3 Isovolume smoothing

With the publication of Andrew Kuprat [6], we have implemented many algorithm of isovolume smoothing. In one hand, a 2D algorithm which smoothes plane curves. In the other hand, a 3D algorithm for surface meshes. Our results are shown on figures (3) and (4).

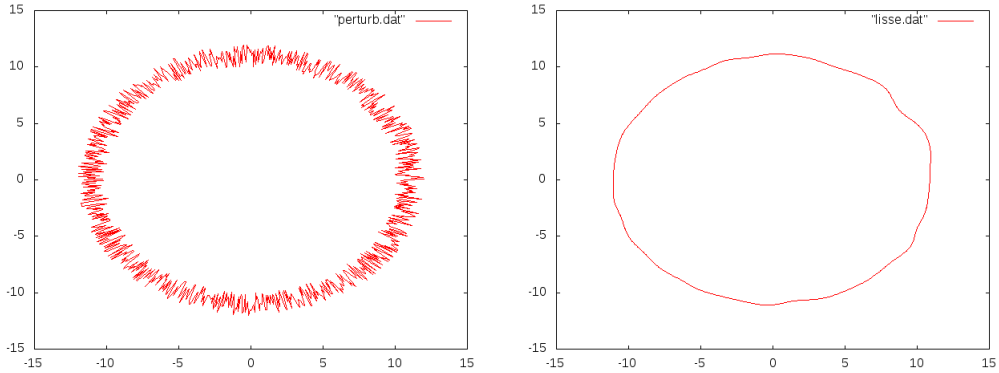


FIGURE 3 – Initial condition and the 10000 iterations result of the plane curve smoothing algorithm.

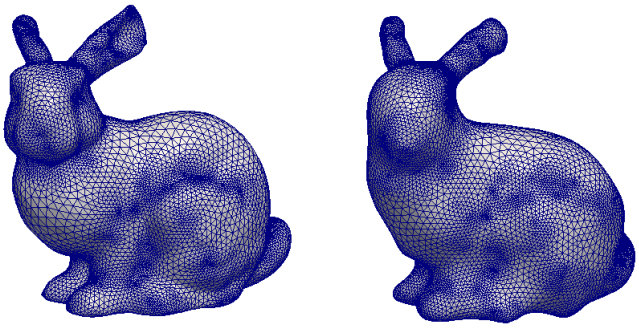


FIGURE 4 – Initial condition and the 12 iterations result of the 3D smoothing algorithm.

1.4 Conclusion

Although *CGAL* offers a lot of geometric tools, its use is unsatisfying. We should program some algorithms to complete its skills. All the expected features were implemented with success.

2 Introduction

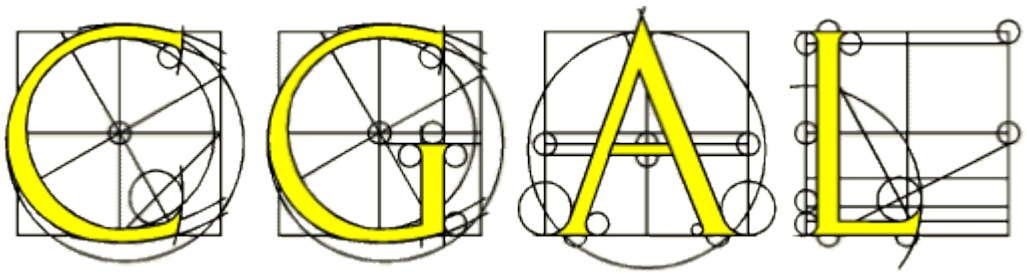
Dans les réacteurs chimiques industriels, un très grand nombre de particules de toutes les tailles et de toutes les formes est transporté à travers l'écoulement. Parallèlement, l'atomisation du fluide créée en sortie d'un injecteur diesel, comme présenté figure (5), forme une miriade de particules ayant chacune leur propre trajectoire. Il en va de même pour l'affinage de l'acier, qui est l'opération consistant à injecter de l'argon dans l'acier liquide afin de récupérer toutes les impuretés. Utiliser la simulation numérique pour caractériser la forme des particules, leur nombre et leur vitesse demande une modélisation interfaciale extrêmement précise. De plus, de nombreux phénomènes se produisent à l'approche de deux interfaces, les chocs ou la coalescence en sont deux exemples. Plusieurs méthodes numériques ont été développées pour modéliser ces interfaces. Elles se classent en deux catégories. La première regroupe les méthodes eulériennes de type *level-set* ou *VOF*. On définit dans chaque maille une fonction couleur que l'on advecte dans l'écoulement, l'interface est recréée par une isosurface. Cette méthode a pour inconvénient de requérir un grand nombre de mailles du domaine fluide pour être précise, en contrepartie, la topologie des objets peut être grossière. La seconde catégorie regroupe les méthodes lagrangiennes, il s'agit cette fois de représenter l'interface par un maillage, ce qui implique que la topologie des objets doit être beaucoup plus précise. En contrepartie, les mailles du domaine fluide peuvent être plus grossières. Encore une fois, la simulation numérique est limitée par la capacité des machines, le progrès de ces dernières permet le développement de nouvelles techniques ou bien l'utilisation de techniques écartées à une époque où la technologie ne permettait pas leur mise en oeuvre.

Ce TER est le travail préliminaire à une méthode lagrangienne de représentation des interfaces. Notre travail consiste à programmer des outils permettant la manipulation de maillages, comme la fusion et le raffinement, et le calcul de certaines de leurs propriétés intrinsèques, comme la courbure ou le volume dans le cas d'un maillage surfacique fermé. Notre travail s'est organisé en deux parties, la première étant la recherche et l'essai de bibliothèques préexistantes, comme CGAL. Et la seconde partie à programmer des algorithmes publiés dans des revues scientifiques de qualité.

FIGURE 5 – Visualisation instantanée de l'interface liquide/gaz dans le cas DNS.

Dans la suite de ce rapport, nous appelerons "qualité du maillage" sa qualité de surface, c'est-à-dire une courbure cohérente avec sa géométrie globale, ainsi qu'une forme de maillage "adaptée" à la géométrie.

3 Bibliothèque CGAL



3.1 Introduction

La bibliothèque CGAL (Computational Geometry Algorithms Library) est une bibliothèque développée en C++ par l'INRIA. Cette bibliothèque offre quantité d'outils géométriques, comme la génération de maillages, algorithmes de triangulations, diagramme de Voronoï, recherche du plus proche élément, union, différence, intersection de maillages, enveloppe convexe ... Une documentation complète et détaillée est disponible en ligne sur http ://www.cgal.org [1] . Dans notre TER, nous avons utilisé la version 3.5.1 de la bibliothèque sous Ubuntu (Version 9.04, 9.10 et 10.04).

Ce qui nous a amené à découvrir CGAL est son arsenal de techniques de raffinement de maillages surfaciques. Nous espérons trouver une méthode permettant de raffiner le maillage, d'améliorer sa qualité tout en conservant son volume.

3.2 Représentation d'un maillage surfacique dans CGAL

Le maillage surfacique dans CGAL est représenté dans une structure appelée *Polyhedron* (figure 6).

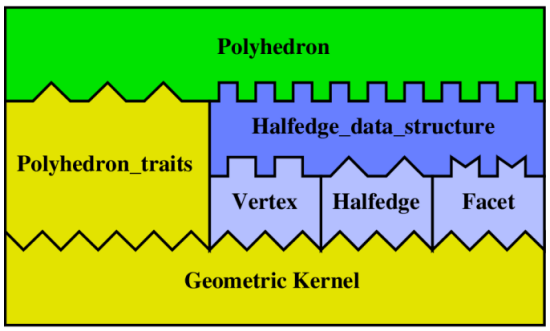


FIGURE 6 – Structure d'un *Polyhedron*.

Tout d'abord, nous devons spécifier à CGAL le système de coordonnées *Geometric Kernel*. La bibliothèque offre plusieurs possibilités : cartésien 2D ou 3D, simple ou double précision ... (Voir [1] pour plus de précisions)
Le *polyhedron* est un ensemble de *vertices* reliés entre eux par des *halfedges*. Les demi-arêtes représente une arête dans une face donnée. Les *facets* sont constitués de *halfedges*. La figure (7) représente cette structure.

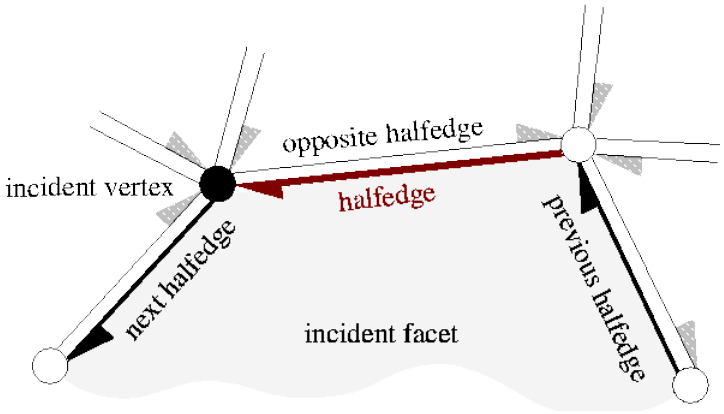


FIGURE 7 – Illustration de la structure de *Polyhedron* utilisant les *halfedges*

Chaque demi-arête pointe vers sa demi-arête suivante, sa précédente, son opposée, ainsi que vers le sommet vers lequel elle est dirigée et sa facette associée. Réciproquement, tout sommet pointe vers une demi-arête et toute face pointe vers une de ses demi-arêtes. Dans une face, les demi-arêtes sont reliées dans un même sens de circulation qui définira l'orientation de la face. Cette structure permet de faire aisément deux opérations élémentaires : la circulation de points autour d'une face et la circulation de points autour d'un sommet (voir figures 8 et 9).

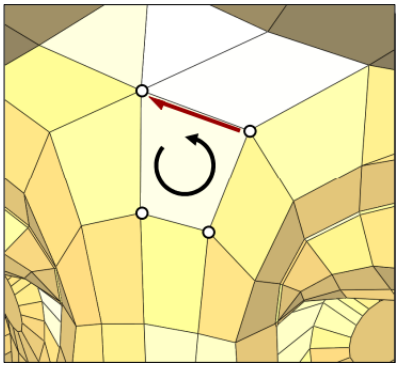


FIGURE 8 – Circulation de points autour d'une face. Prenons une demi-arête d'une face, la suivante est obtenue en utilisant la demi-arête suivante pointée par la demi-arête. Les points sont obtenus grâce aux demi-arêtes.

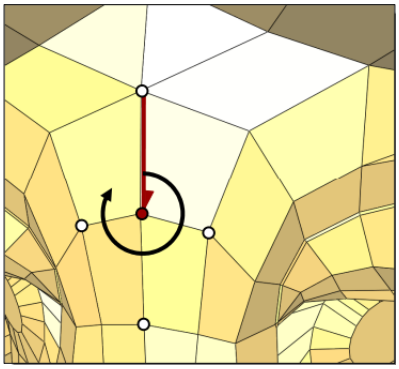


FIGURE 9 – Circulation de points autour d'un sommet. Prenons une demi-arête pointée vers un somment, la suivante est obtenue en prenant l'opposée de la demi-arête suivante. Les points sont obtenus en prenant le point de la demi-arête opposée.

3.3 Subdivision

Dans cette section, nous allons explorer quelques méthodes de subdivision implémentées dans CGAL. Nous prêterons attention à l'amélioration de qualité de surface obtenue ainsi que la différence de volume. Nous définissons également le lissage d'une surface, qui est l'opération consistant à supprimer les parasites de courbure locale.

3.3.1 Catmull-Clark

Cet algorithme présente un inconvénient pour nous, le maillage résultant est composé de quadrangles. Or nous voulons obtenir un maillage résultant triangulaire. Étant donné ce problème, nous allons présenter succintement cet algorithme par la figure (10).

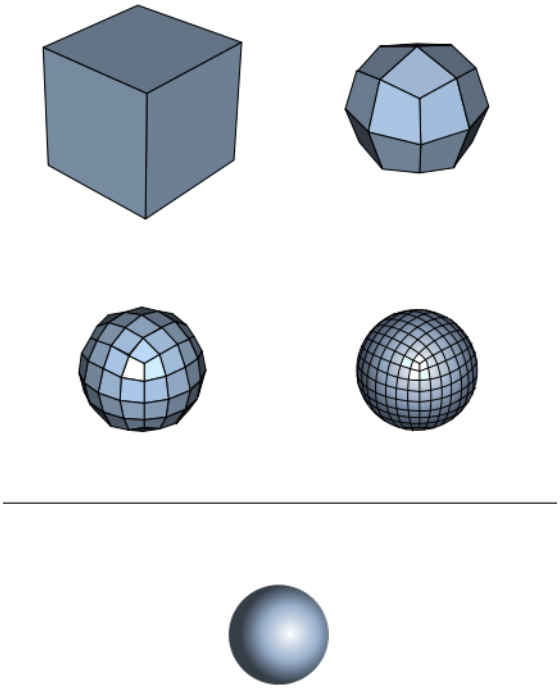


FIGURE 10 – Algorithme de Catmull-Clark pour un cube. Les trois premiers pas sont représentés ainsi que la surface vers laquelle l'algorithme converge.

La qualité de surface obtenue à la fin du raffinement est très bonne, cependant, il est très clair que le volume final est différent du volume initial.

3.3.2 1-to-4-subdivision

La figure (11) présente l'algorithme.

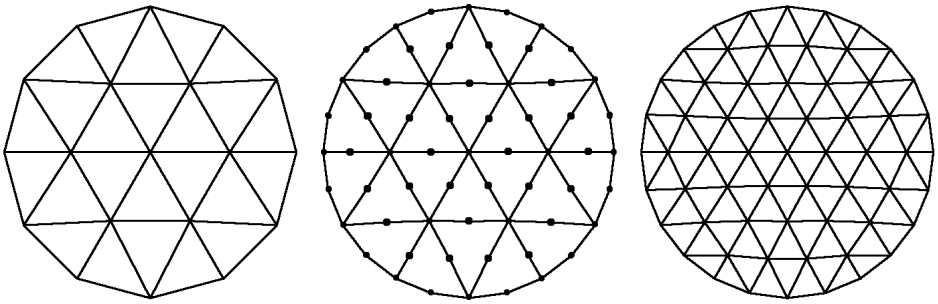


FIGURE 11 – Présentation d'une itération de l'algorithme 1-to-4-subdivision. (a) Situation initiale. (b) Calcul du centre des arêtes. (c) Les nouveaux centres sont reliés entre eux dans chaque face.

L'algorithme ne présente aucun intérêt dans le lissage de la surface, il se contente juste de subdiviser les faces, par conséquent, il n'y a aucune perte de volume.

3.3.3 $\sqrt{3}$-subdivision

La figure (12) présente l'algorithme.

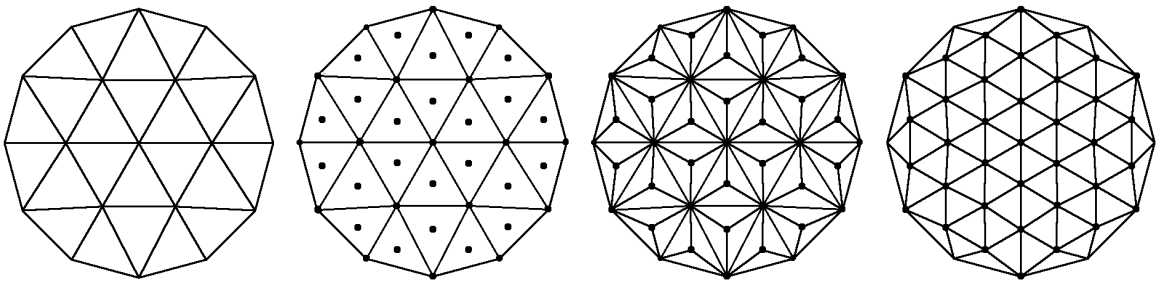


FIGURE 12 – Présentation d'une itération de l'algorithme $\sqrt{3}$-subdivision. (a) Situation initiale. (b) Calcul du centre des faces. (c) Les centres sont reliés aux anciens noeuds des faces. (d) Permutation de toutes les arêtes initiales.

Au niveau de la qualité de la surface obtenue, d'après le document [3], la surface converge vers une surface de classe $\mathbf{C^2}$. En revanche, le volume n'est pas conservé comme le montre la figure (13).

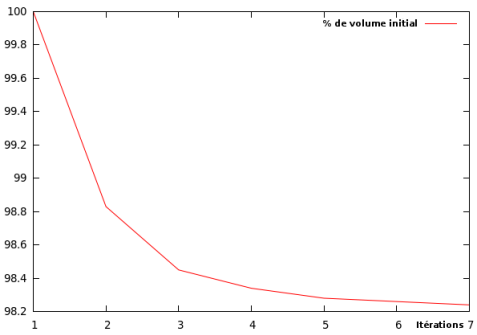


FIGURE 13 – Graphe montrant la décroissance du volume total en fonction du nombre d'itération de l'algorithme $\sqrt{3}$-subdivision. Ce test a été effectué sur le lapin de Stanford présenté en annexe section A.3.

3.4 Union, intersection et différence de maillages

L'une des problèmatiques de notre TER était la fusion de deux maillages à volume constant. La bibliothèque CGAL offre une palette d'outils permettant d'effectuer les opérations d'union, d'intersection et de différence entre plusieurs maillages. La structure encapsulant ces opérations porte le nom de *NefPolyhedron*. Un *NefPolyhedron* peut se construire à partir d'un *Polyhedron*. Les opérateurs $+$, $-$ et $*$ ont été définis pour permettre les opérations qui sont respectivement l'union, la différence et l'intersection. L'objet retourné est un *NefPolyhedron*. Les opérateur $+=$, $-=$ et $*=$ ont également été définis. Le listing (1), présenté en annexe page 26, permet d'effectuer l'opération de différence entre deux *NefPolyhedron*.
Nous avons testé cette opération avec un cube et une sphère, le résultat est présenté figure (14). Nous pouvons noter que le cube a été entièrement remaillé, mais de manière peu élégante. En effet, les triangles obtenus sont très fins et très allongés. Effectuer une opération de lissage sur un tel objet s'avérerait catastrophique. Il faudrait au préalable le remailler.
La raison de ce résultat est que le nombre de mailles de la sphère est très grand par rapport à celui du cube et que la taille des mailles est petite par rapport à celles des mailles du cube. Pour obtenir des résultats acceptables, nous préconisons d'utiliser des maillages de qualité identique.

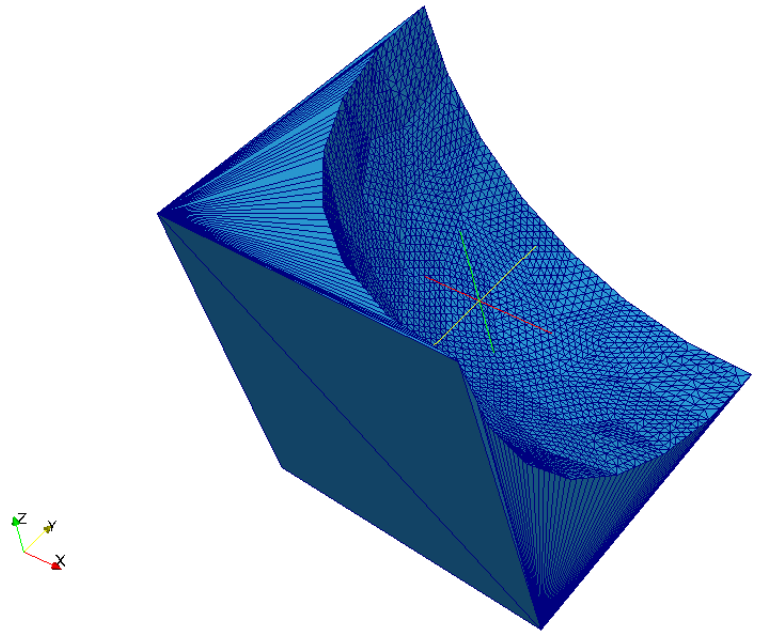


FIGURE 14 – Différence du cube par la sphère (objets présentés en annexe dans la section (A.3) page 25)

Quant à la problématique de la fusion de deux maillages à volume constant, hors contexte physique, elle est difficile à définir. Volume constant signifie que le volume final est égal à la somme des volumes des composants. Nous proposons, pour illustrer les capacités de CGAL et de la faisabilité de cette opération, de programmer un algorithme permettant cette opération. Cet algorithme consiste à approcher deux sphères suffisament proche pour qu'elles s'intersectent juste assez pour permettre l'union des maillages. Grâce à la formule (13), nous calculons le volume des deux sphères. Nous appliquons l'algorithme de fusion de CGAL, puis nous récupérons le volume perdu en effectuant un déplacement normal des noeud.

4 Advection de maillage

Le but de cette partie est d'advecter un maillage dans un champ vectoriel sans perte d'information. En effet, lors de cette opération, il se peut que des triangles composant le maillage deviennent très élancés (une longueur très grande devant les autres). Ceci peut créer une perte ou un gain de volume, ce qui peut s'avérer gênant dans le cadre d'un champ à divergence nulle. Pour illustrer notre propos, considérons le champs vectoriel à divergence nulle (1) provenant de [4] :

$$\begin{cases} x &= 2\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z)\cos(\pi t/T) \\ y &= -\sin(2\pi x)\sin^2(\pi y)\sin(2\pi z)\cos(\pi t/T) \\ z &= -\sin(2\pi x)\sin(2\pi y)\sin^2(\pi z)\cos(\pi t/T) \end{cases} \tag{1}$$

À l'aide du schéma Runge-Kutta 4, nous avons advecté le maillage *sphereSimple* décrit en annexe page 25. Pour $T = 3$, un temps total de 6.0 et un pas de temps de 0.01 (ce qui correspond à une période entière du cosinus en temps), nous obtenons la variation de volume décrite sur la figure (15).

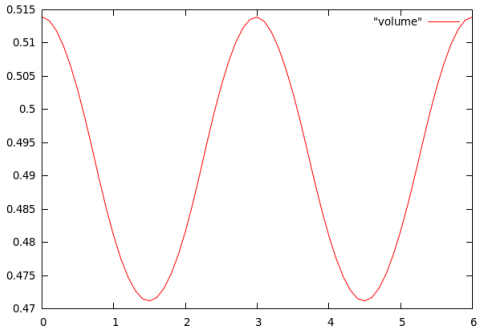


FIGURE 15 – Variation de volume en fonction du temps du maillage *sphereSimple* par advection dans le champ (1).

Afin de minimiser l'impact de l'advection sur la variation de volume, nous proposons de raffiner localement le maillage. C'est-à-dire, là où les triangles sont "trop grands", appliquer un algorithme de subdivision. Les deux algorithmes présentés dans la section 3.3 ne sont pas adaptés car ils impliquent un raffinement global. Ce que nous proposons alors, c'est d'appliquer l'algorithme $\sqrt{3}$-*subdivision* jusque à l'étape (c), c'est-à-dire, sans la permutation des arêtes.
Nous pouvons noter que l'utilisation d'un maillage très fin minimise cet effet.

5 Courbure de maillage

Estimer la qualité d'un maillage implique le choix de critères de qualité. La courbure est un élément pertinent. Nous proposons d'utiliser l'algorithme de courbure décrit par *Rao V. Garimella* et *Blair K. Swartz* dans leur article [5].
Le principe est d'approximer localement le maillage par une quadrique d'équation $z = ax^2 + bxy + cy^2$. Les courbures principales, la courbures de Gauss et la courbure moyenne étant données par les équations (2).

$$\begin{cases} \kappa_1 &= a + c + \sqrt{(a-c)^2 + b^2} \\ \kappa_2 &= a + c - \sqrt{(a-c)^2 + b^2} \\ K &= 4ac - b^2 \\ H &= a + c \end{cases} \tag{2}$$

Soit (X, Y, Z) le repère global. Voici l'algorithme, décrit pas à pas, permettant de calculer la courbure en un point du maillage :

1. Soit $\mathbf{p}$ un point du maillage. Nous devons estimer la normale $\mathbf{n}$ en ce point. Pour ce faire, nous allons utiliser la moyenne des normales des faces adjacentes au point, pondérées par la surface de la face sur la surface totale du voisinage.
2. Définissons un repère local (X', Y', Z') dont l'origine est en $\mathbf{p}$. Prenons Z' colinéaire avec la normale $\mathbf{n}$. Si Z' n'est pas colinéaire à X, nous prenons X' égal à la projection de X sur le plan tangent définit par $\mathbf{n}$. Dans le cas contraire, nous prendrons X' égal à la projection de Y. La coodonnée Y' s'obtient par le produit vectoriel de Z' par X'.
3. Considérons un ensemble de points voisins de $\mathbf{p}$. Dans notre implémentation, nous avons pris l'ensemble des points directement voisin de $\mathbf{p}$.
4. Effectuons un changement de repére des points, du repère global au repère local.
5. Calculons les coefficient de la quadrique par la méthode des moindres carrés :

$$\begin{bmatrix} x_1^2 & x_1 y_1 & y_1^2 \\ \vdots & \vdots & \vdots \\ x_n^2 & x_n y_n & y_n^2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix}$$

   Nous rappelons que la méthode des moindres carrés consiste à trouver une solution minimale du système $\mathbf{Ax} = \mathbf{b}$ dans le sens que $\mathbf{x}$ soit la solution de $\mathbf{A^T Ax} = \mathbf{A^T b}$, le système doit être de plus surdéterminé pour assurer l'unicité de la solution. Ce qui revient ici à résoudre un système 3x3. Dans notre implémentation, nous avons utilisé un gradient conjugué. La méthode de Cramer s'était avérée moins précise.
6. Estimation de la courbure via la formule (2).

6 Lissage de maillages à volume constant

6.1 Introduction

Nous allons présenter ici un ensemble d'algorithmes de lissage de courbes planes, d'une part, et d'autre part de lissage de maillages surfaciques 3D. Ces algorithmes proviennent de la publication de *Andrew Kuprat* [6]. Pour implémenter ces algorithmes en *F0RTRAN 90*, pour le cas de la 3D, nous nous sommes inspiré de la structure de donnée du *Polyhedron* décrite dans la section (3.2).

6.2 Algorithme 1 : Lissage simple de courbes 2D

On se donne une courbe fermée (voir figure (16)) formée de n points $(x_0, x_1, ..., x_{n-1}, x_n = x_0)$ reliés par des segments.

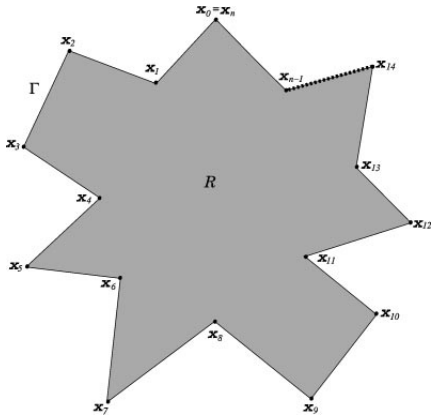


FIGURE 16 – Courbe fermée formée de n points reliés par des segments.

Le principe est de déplacer le point x_1 parallèlement au segment $[x_1, x_2]$ pour le mettre au niveau du milieu du segment comme montré dans la figure (17).

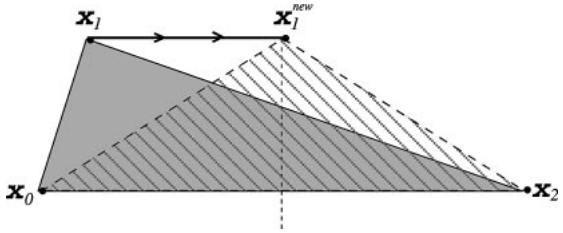


FIGURE 17 – Principe de l'algorithme 1, déplacement horizontal du noeud central.

Pour un vecteur $\mathbf{v}(x, y)$, on définit $\mathbf{v}^\perp = (-y, x)$.
Nous obtenons ainsi le l'algorithme (1) de lissage.

Algorithm 1 Lissage simple de courbes 2D

Pour $i = 1$ à $n - 1$ faire

Modifier la position de x_i en fonction de x_{i-1} et de x_{i+1}

$$\hat{\mathbf{n}} \leftarrow \frac{(x_{i+1} - x_{i-1})^\perp}{||\,(x_i - x_{i-1})^\perp\,||}$$
$$A \leftarrow \frac{1}{2}\,(x_{i+1} - x_{i-1})^\perp\,(x_i - x_{i-1})$$
$$h \leftarrow 2\frac{A}{||\,(x_i - x_{i-1})^\perp\,||}$$
$$x_{i+1} \leftarrow \frac{1}{2}\,(x_{i-1} + x_{i+1}) + h\hat{\mathbf{n}}$$

Cet algorithme écrit en *FORTRAN 90* a été testé sur un cercle perturbé (voir figure (18)). Les perturbations ont été faites à l'aide d'un générateur de nombres aléatoires.

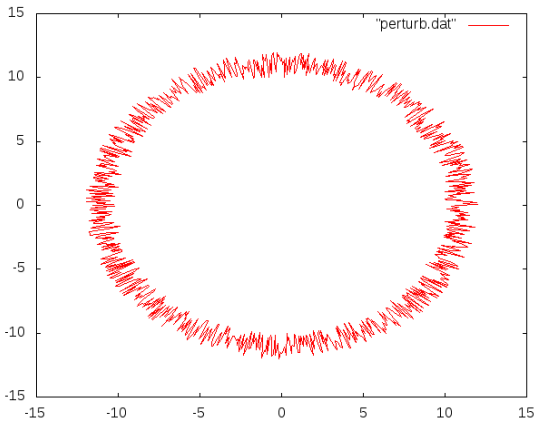


FIGURE 18 – Cercle perturbé.

Après 100 itérations, on obtient un cercle moins perturbé, comme le montre la figure (19).

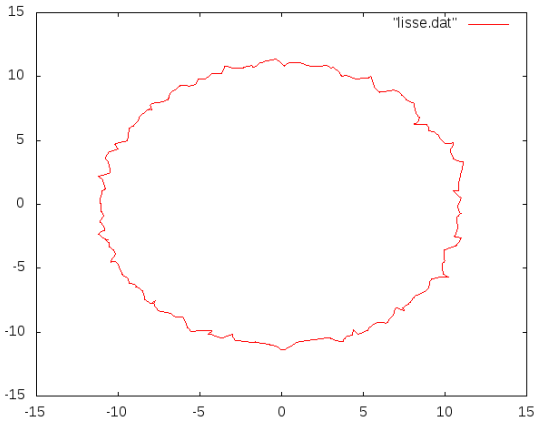


FIGURE 19 – Résultat après 100 itérations de l'algorithme 1.

Après 10000 itérations, on obtient un assez bon cercle, comme on peut l'observer figure (20).

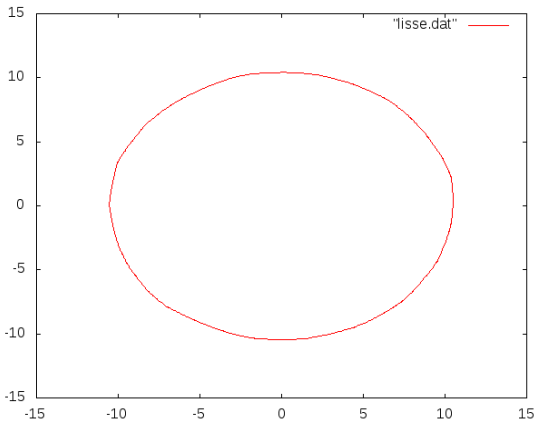


FIGURE 20 – Résultat après 10000 itérations de l'algorithme 1.

Bien que cet algorithme fonctionne, il peut être amélioré.

6.3 Algorithme 2 : Lissage double de courbe 2D

Cette fois-ci, on modifie deux points au lieu d'un seul en fonction de leurs voisins.

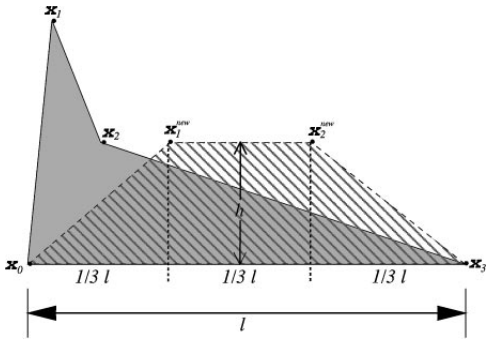


FIGURE 21 – Principe du second algorithme. Les points sont traités par paire, ils sont déplacés normalement et tangentiellement.

Voici l'algorithme (2) :

Algorithm 2 Lissage double de courbes 2D

Pour $i = 1$ à $n - 1$ faire

Modifier la position de x_{i+1} et de x_{i+2} en fonction de x_i et de x_{i+3}

$$\hat{\mathbf{n}} \leftarrow \frac{(x_{i+3} - x_i)^{\perp}}{||\,(x_{i+3} - x_i)^{\perp}\,||}$$
$$A \leftarrow \frac{1}{2}(x_{i+3} - x_i)^{\perp}(x_{i+2} - x_i) + \frac{1}{2}(x_{i+2} - x_i)^{\perp}(x_{i+1} - x_i)$$
$$h \leftarrow \frac{3}{2}\frac{A}{||\,(x_{i+3} - x_i)^{\perp}\,||}$$
$$x_{i+1} \leftarrow \frac{2}{3}x_i + \frac{1}{3}x_{i+3} + h\hat{\mathbf{n}}$$
$$x_{i+2} \leftarrow \frac{1}{3}x_i + \frac{2}{3}x_{i+3} + h\hat{\mathbf{n}}$$

Testons maintenant cet algorithme sur notre cercle perturbé figure (18).

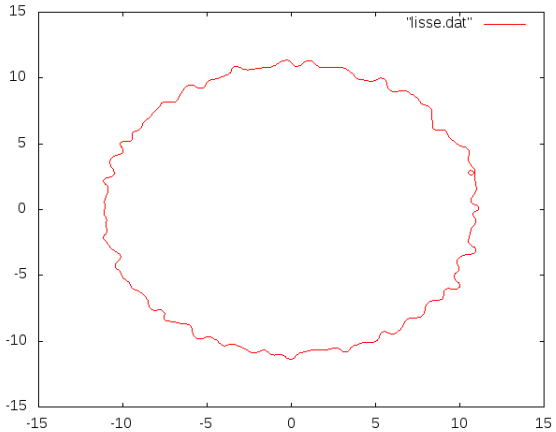


FIGURE 22 – Test de l'algorithme 2 pour 100 itérations

Pour 100 itérations (figure 22), le cercle est plus lisse que dans l'algorithme précédent, donc l'algorithme 2 semble être le plus efficace. En revanche, pour 10000 itérations (figure 23), on remarque

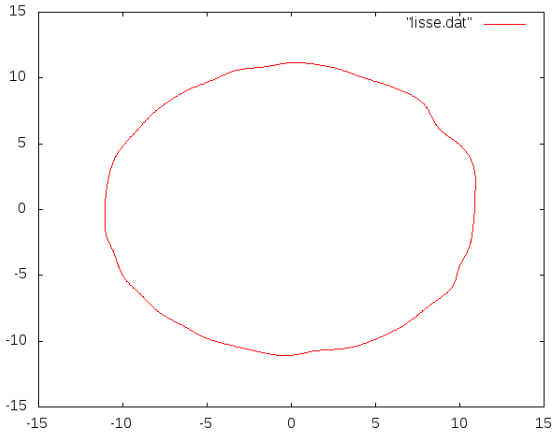


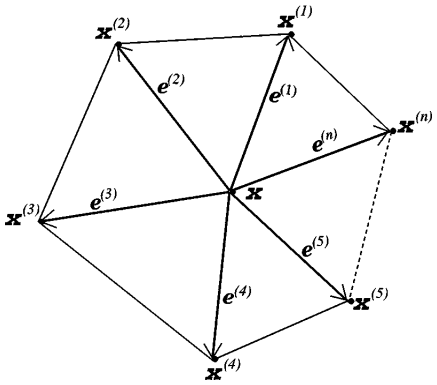
FIGURE 23 – Test de l'algorithme 2 pour 10000 itérations

que le cercle possède quelques bosses, même s'il est très lissé. Cela est dû au fait que l'algorithme 2 conserve davantage la géométrie initiale de la courbe de départ.

NB : Ces algorithmes fonctionnent également sur des courbes non fermées. Le principe est d'appliquer l'algorithme en tout point, sauf aux extrémités.

6.4 Algorithme 3 : Lissage simple d'un maillage surfacique 3D

On considère cette fois une surface fermée constituée de triangles. Soit un sommet $\mathbf{x}$. Notons $\mathbf{x}^{(j)}$ pour $j = 1$ à n les n voisins de $\mathbf{x}$. On définit les vecteurs $\mathbf{e}^{(j)}$ tels que $\mathbf{e}^{(j)} = \mathbf{x}^{(j)}$ - $\mathbf{x}$.



L'algorithme de lissage d'un maillage surfacique 3D est présenté en (3).

Algorithm 3 Lissage simple de courbes 3D

Pour chaque sommet $\mathbf{x}$ entouré par les sommets $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(n)}\}$

$$\hat{\mathbf{n}} \leftarrow \frac{\sum_{j=1}^{n} \mathbf{e}^{(j)} \times \mathbf{e}^{(j+1)}}{||\sum_{j=1}^{n} \mathbf{e}^{(j)} \times \mathbf{e}^{(j+1)}||}$$
$$d\mathbf{x}^s \leftarrow \frac{1}{n} \sum_{j=1}^{n} \mathbf{x}^{(j)}$$
$$\mathbf{x} \leftarrow \mathbf{x} + d\mathbf{x}^s - (d\mathbf{x}^s|\hat{\mathbf{n}})\hat{\mathbf{n}}$$

6.5 Algorithme 4 : Lissage double de maillages surfaciques en 3D

De la même façon que pour les courbes, on améliore l'algorithme de lissage en modifiant 2 points en fonction de leurs voisins.

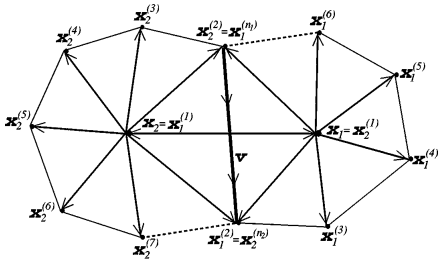


FIGURE 24 – Notations pour la modification des sommets $\mathbf{x_1}$ et $\mathbf{x_2}$

Voici donc l'algorithme :

Algorithm 4 Lissage double de maillages surfaciques en 3D

Pour chaque arête $\overline{\mathbf{x_1}\mathbf{x_2}}$ entourée par les sommets $\left(\mathbf{x}_i^{(j)}\right)_{i=1,2}^{j=1,\ldots,n_i}$

$A_i \leftarrow \sum_{j=1}^{n} \mathbf{e}^{(j)} \times \mathbf{e}^{(j+1)}$
$\mathbf{v} \leftarrow \mathbf{e}_2^{(n_2)} - \mathbf{e}_2^{(2)}$
$\mathbf{x}_1^s \leftarrow \dfrac{1}{n_1 n_2 - 1}\left(\sum_{j=2}^{n_2} \mathbf{x}_2^{(j)} + n_2 \sum_{j=2}^{n_1} \mathbf{x}_1^{(j)}\right)$
$\dfrac{1}{n_2}\left(\mathbf{x}_1^s + \sum_{j=2}^{n_2} \mathbf{x}_2^{(j)}\right)$
$d\mathbf{x}_i^s \leftarrow \omega\,(\mathbf{x}_i^s - \mathbf{x}_i),\ i = 1, 2,\ 0 < \omega \leq 1$
$\mathbf{A} \leftarrow \mathbf{A}_1 + \mathbf{A}_2 - \mathbf{v} \times (d\mathbf{x}_1^s - d\mathbf{x}_2^s)$
if $(||\mathbf{A}|| > \varepsilon)$ **then**
  $\hat{\mathbf{n}} \leftarrow \dfrac{\mathbf{A}}{||\mathbf{A}||}$
  $\hat{\mathbf{n}} \leftarrow -\dfrac{(d\mathbf{x}_1^s \cdot \mathbf{A}_1 + d\mathbf{x}_2^s \cdot \mathbf{A}_2 + d\mathbf{x}_2^s \cdot \mathbf{v} \times d\mathbf{x}_1^s)}{||\mathbf{A}||}$
  $\mathbf{x}_i \leftarrow \mathbf{x}_i + d\mathbf{x}_i^s + h\hat{\mathbf{n}}, i = 1, 2$
end if

ω et ε sont des paramètres donnés dans l'algorithme. ε est un petit nombre, proche de l'erreur machine. ω sert à limiter les changements lors d'une itération. Pour $\omega = 1$, les changements sont maximaux alors que pour ω proche de zéro, les changements sont amoindris donc il faut plus d'itérations pour obtenir le même lissage.

Une application de cet algorithme est présenté figure (25).

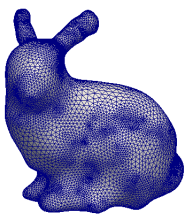


FIGURE 25 – Lapin de Stanford après 400 itérations de l'algorithme 4 avec $\omega = 1.0$ et $\varepsilon = 10^{-14}$.

7 Calculs théoriques

7.1 Calcul de la surface d'une courbe plane

Théorème 7.1. *Soit $C = (\Gamma_i)_{i\in\{0,\dots,N\}}$ une courbe fermée du plan composée de $N+1$ segments. Les sommets des segments sont notés $(\mathbf{P_i} = (x_i, y_i))_{i\in\{0,\dots,N\}}$ et $\mathbf{P_i}$ et $\mathbf{P_{i+1}}$ sont les sommets de Γ_i. La surface est orientée, les points $\mathbf{P_i}$ seront donc définis dans le sens trigonométrique pour une aire positive. La surface S délimitée par la courbe C est donnée par la formule*

$$S = \frac{1}{2}\sum_{k=0}^{N}(x_k)(y_{k+1} - y_{k-1}) \tag{3}$$

Démonstration.

$$\exists\ \mathbf{F} : \mathbb{R}^2 \longrightarrow \mathbb{R}^2\ S = \int_S dS = \int_C \nabla\cdot\mathbf{F}ds \tag{4}$$

On veut $\nabla\cdot\mathbf{F} = 1$. Prenons $\mathbf{F} : (x, y) \mapsto (x, 0)$.

$$\int_S dS = \int_C \mathbf{F}\cdot\mathbf{n}ds = \sum_{k=0}^{N}\int_{\Gamma_k}\mathbf{F}\cdot\mathbf{n_k}ds = \sum_{k=0}^{N}(\mathbf{n_k}\cdot\mathbf{e_x})\int_{\Gamma_k} xds \tag{5}$$

On pose $\mathbf{D_i} = \mathbf{P_{i+1}} - \mathbf{P_i}$.
Soit $\mathbf{P} \in \Gamma_i$, $\mathbf{P} = \mathbf{P_i} + u\mathbf{D}$ avec $u \in [0, 1]$.

$$\begin{aligned}\mathbf{P}(u) &= (x(u), y(u)) \\ &= (x_i + \alpha_x u, y_i + \alpha_y u)\end{aligned} \tag{6}$$

avec $\alpha_x = x_{i+1} - x_i$ et $\alpha_y = y_{i+1} - y_i$

Changement de variable :

$$\int_{\Gamma_k} xds = \int_0^1 x(u)Jdu \tag{7}$$

avec :

$$\begin{aligned}J &= \left\|\frac{\partial\mathbf{P}}{\partial u}\right\| \\ &= \|\mathbf{D}\|\end{aligned} \tag{8}$$

De plus $\mathbf{n_k} = \dfrac{\mathbf{D}\times\mathbf{e_z}}{\|\mathbf{D}\|}$

$$(\mathbf{n_k}\cdot\mathbf{e_x})\int_{\Gamma_k} xds = ((\mathbf{D}\times\mathbf{e_z})\cdot\mathbf{e_x})\int_0^1 (x_k + \alpha_x u)du \tag{9}$$

$$= (y_{k+1} - y_k)\frac{x_i + x_{i+1}}{2} \tag{10}$$

d'où :

$$S = \frac{1}{2}\sum_{k=0}^{N}(y_{k+1} - y_k)(x_k + x_{k+1}) \tag{11}$$

avec $x_{N+1} = x_0$ et $y_{N+1} = y_0$.

On peut cependant mieux faire :

$$
\begin{aligned}
S &= \frac{1}{2}\sum_{k=0}^{N}(y_{k+1}-y_k)(x_k+x_{k+1}) \\
&= \frac{1}{2}\sum_{k=0}^{N}(y_{k+1}-y_k)(x_{i+1})+\sum_{k=0}^{N}(y_{k+1}-y_k)(x_k) \\
&= \frac{1}{2}\sum_{k=0}^{N}(y_{k+1}-y_k)(x_{i+1})+\sum_{k=0}^{N}(y_{k+2}-y_{k+1})(x_{k+1}) \\
&= \frac{1}{2}\sum_{k=0}^{N}(y_{k+1}-y_k+y_{k+2}-y_{k+1})(x_{k+1}) \\
&= \frac{1}{2}\sum_{k=0}^{N}(y_{k+1}-y_{k-1})(x_k) \qquad (12)
\end{aligned}
$$

□

7.2 Calcul de volume d'un maillage triangulaire surfacique 3D

Théorème 7.2. *Soit $S=(\tau_i)_{i\in\{0,\dots,N\}}$ une surface fermée composée de $N+1$ triangles de $\mathbb{R}^3$. On note $\{P_0^k,P_1^k,P_2^k\}$ les points du triangle τ_k, avec (x_i^k,y_i^k,z_i^k) les coordonnées du point P_i^k. Le volume V délimité par S est donné par la formule*

$$V=\frac{1}{6}\sum_{k=0}^{N}((y_1^k-y_0^k)(z_2^k-z_0^k)-(y_2^k-y_0^k)(z_1^k-z_0^k))(x_0^k+x_1^k+x_2^k) \qquad (13)$$

Démonstration.

$$\exists\ \mathbf{F}:\mathbb{R}^3\longrightarrow\mathbb{R}^3\ V=\int_V dV=\int_V \nabla\cdot\mathbf{F}dV \qquad (14)$$

On veut $\nabla\cdot\mathbf{F}=1$. Prenons $\mathbf{F}:(x,y,z)\mapsto(x,0,0)$.

$$\int_V dV=\int_S \mathbf{F}\cdot\mathbf{n}dS=\sum_{k=0}^{N}\int_{\tau_k}\mathbf{F}\cdot\mathbf{n_k}dS=\sum_{k=0}^{N}(\mathbf{n_k}\cdot\mathbf{e_x})\int_{\tau_k}xdS \qquad (15)$$

On pose $\mathbf{C_1}=P_1-P_0$ et $\mathbf{C_2}=P_2-P_0$.
Soit P un point de τ_k, $P=P_0+u\mathbf{C_1}+v\mathbf{C_2}$ avec $u\geq 0$, $v\geq 0$ et $u+v\leq 1$.

$$
\begin{aligned}
\mathbf{P}(u,v) &= (x(u,v),y(u,v),z(u,v)) \\
&= (x_0+\alpha_x u+\beta_x v, y_0+\alpha_y u+\beta_y v, z_0+\alpha_z u+\beta_z v) \qquad (16)
\end{aligned}
$$

avec $\alpha_x=x_1-x_0$ et $\beta_x=x_2-x_0$.

Changement de variable :

$$\int_{\tau_k}xdS=\int_0^1\int_0^{1-v}x(u,v)Jdudv \qquad (17)$$

avec :

$$
\begin{aligned}
J &= \left\|\frac{\partial\mathbf{P}}{\partial u}\times\frac{\partial\mathbf{P}}{\partial v}\right\| \\
&= \|\mathbf{C_1}\times\mathbf{C_2}\| \qquad (18)
\end{aligned}
$$

De plus $\mathbf{n_k}=\dfrac{\mathbf{C_1}\times\mathbf{C_2}}{\|\mathbf{C_1}\times\mathbf{C_2}\|}$

$$(\mathbf{n_k} \cdot \mathbf{e_x}) \int_{\tau_k} x dS = ((\mathbf{C_1} \times \mathbf{C_2}) \cdot \mathbf{e_x}) \int_0^1 \int_0^{1-v} (x_0 + \alpha_x u + \beta_x v) du dv \quad (19)$$

$$= ((y_1^k - y_0^k)(z_2^k - z_0^k) - (y_2^k - y_0^k)(z_1^k - z_0^k)) \frac{x_0^k + x_1^k + x_2^k}{6} \quad (20)$$

D'où l'équation (13).

□

8 Conclusion

En premier lieu, la bibliothèque CGAL nous a apporté les outils nécessaires à la fusion de maillages, elle nous a également inspiré une représentation de maillage adaptée à de nombreux algorithmes. Malheureusement, elle n'offrait aucun algorithme de lissage à volume constant, ni d'outils de mesure de courbure (bien qu'intrinsèquement elle utilise la courbure pour certains de ses algorithmes, la version actuelle n'offrait pas d'interface pour y accéder). C'est pour cela qu'il a fallu chercher dans une autre direction, ce qui nous a amené à programmer des algorithmes performants existants publiés dans des revues scientifiques. Le lissage à volume constant s'est révélé très performant, aussi bien en deux dimensions qu'en trois dimensions. La mesure de la courbure quant à elle n'a pas offert des résultats pertinents, bien que dans la publication, cette méthode s'avérait être assez bonne. Pour cette raison nous n'avons pas illustré l'algorithme par un exemple de résultat.
Pour compléter ce TER, les travaux restant à faire sont la coalescence de maillages dans un contexte de mécanique des fluides, et ce, avec un grand nombre d'objet. Pour aller encore plus loin, la détection de collision et réactions.

Annexes

A Maillages

A.1 Formats de maillages

A.1.1 Format *OFF*

Le format *OFF* (*Object File Format*) est le format le plus basique pour représenter un maillage surfacique constitué de polygones.
Le format est le suivant :

OFF
nbSommets nbFaces nbArêtes
x y z
x y z
...
NbSommet v1 v2 v3 ... vN
NbSommet v1 v2 v3 ... vM
...

Exemple : le tétraèdre

OFF
4 4 0
0.0 0.0 0.0
1.0 0.0 0.0
0.0 1.0 0.0
0.0 0.0 1.0
3 0 1 3
3 0 2 1
3 0 3 2
3 1 2 3

A.1.2 Format *VTK*

Le format *VTK* est beaucoup plus complexe que le format *OFF*. Nous allons présenter ici que son utilisation pour représenter un maillage surfacique.
Le format est le suivant :

vtk DataFile Version 1.0
VTK from tetra.off
ASCII

DATASET UNSTRUCTURED_GRID
POINTS *nbPoints* float
x y z
...

CELLS *NbFaces 3*NbFaces*

NbSommets a b c
...

CELL_TYPES *NbFaces*
5 ⟵ 5 est le type pour un triangle
5
...

POINT_DATA *NbPoints*
SCALARS *NomTableValeur* float
LOOKUP_TABLE default
v_1
v_2
...

Exemple : le tétraèdre muni d'un de valeurs à ses sommets

vtk DataFile Version 1.0
VTK from tetra.off
ASCII

DATASET UNSTRUCTURED_GRID
POINTS 4 float
0.0 0.0 0.0
1.0 0.0 0.0
0.0 1.0 0.0
0.0 0.0 1.0

CELLS 4 16
3 0 1 3
3 0 2 1
3 0 3 2
3 1 2 3

CELL_TYPES 4
5
5
5
5

POINT_DATA 4
SCALARS pression float
LOOKUP_TABLE default
-6.1954339628907364
1.0000000000000000
0.0000000000000000
4.0000000000000000

A.1.3 Conversion de format

Beaucoup de maillages sont disponibles sur internet, mais souvent dans un format inadapté à nos besoins. Le format le plus fréquement rencontré est le STL. Le logiciel admesh (disponible sous ubuntu via les dépots habituels, pour les windowsiens le lien est cassé), permet de faire le lien entre le *STL* et le *OFF* via la commande :

admesh –write-off=fichier.off fichier.stl

A.2 Logiciels d'édition de maillage

A.2.1 Gmsh

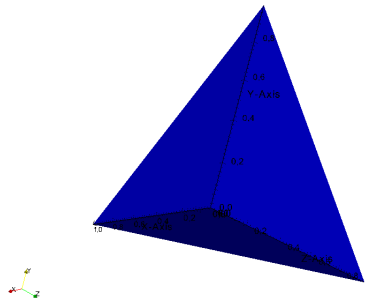
Le logiciel *Gmsh* permet de faire des maillages de qualité. Il implémente le raffinement 1-to-4 décrit section (3.3.2), ce qui nous a été fortement utile avant notre propre implémentation.
Un tutoriel en ligne est disponible sur [2].

A.2.2 Blender

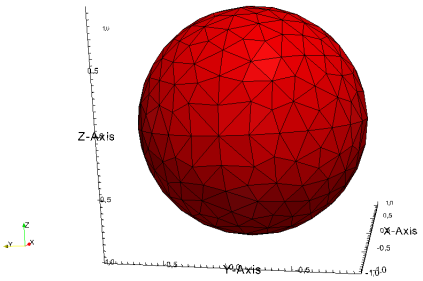
Cette section n'a d'existence que pour mettre en garde le lecteur sur l'utilisation de *Blender*. En effet, bien que sachant manipuler le format *OFF* et *STL*, à l'heure actuelle (version 2.49.2), il ne respecte pas l'orientation des faces. Son utilisation est donc à proscrire dans le calcul scientifique.

A.3 Maillages utilisés pour les tests

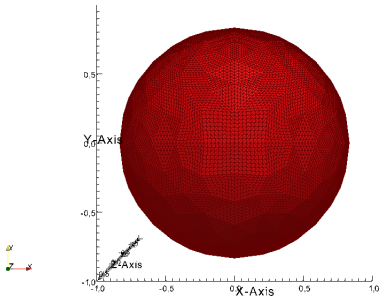
Nom	Tetra
Volume	0.16666
Sommets	4
Arrêtes	6
Faces	4



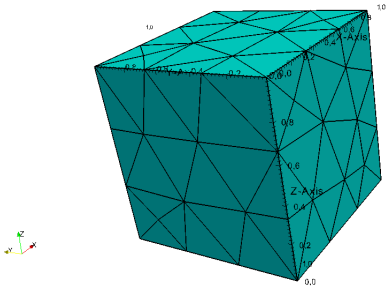
Nom	Sphere Simple
Volume	4.11136
Sommets	326
Arrêtes	972
Faces	648



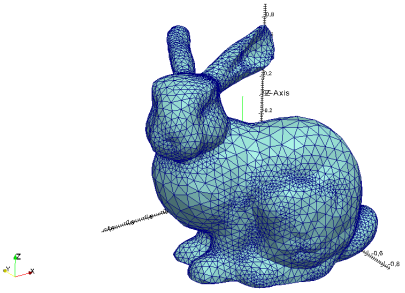
Nom	Sphere
Volume	4.11136
Sommets	20738
Arrêtes	62208
Faces	41472



Nom	Cube
Volume	1.00000
Sommets	68
Arrêtes	198
Faces	132



Nom	Lapin de Stanford
Volume	1.58291
Sommets	4379
Arrêtes	13131
Faces	8754



B Illustrations de code

Listing 1 – Intersection de deux *NefPolyhedron*

```
#include <CGAL/Exact_predicates_exact_constructions_kernel.h>
#include <CGAL/Nef_polyhedron_3.h>
#include <iostream>
#include <fstream>
#include <CGAL/Polyhedron_3.h>
#include <CGAL/IO/Polyhedron_iostream.h>

typedef CGAL::Exact_predicates_exact_constructions_kernel Kernel;
typedef CGAL::Polyhedron_3<Kernel> Polyhedron;
typedef CGAL::Nef_polyhedron_3<Kernel> Nef_polyhedron;

using namespace std;
using namespace CGAL;

int main(int argc, char** argv)
{
    if (argc != 3)
    {
        cout << "Usage: difference mesh1 mesh2" << endl;
        cout << "mesh 1, mesh 2: the input meshes (.off)" << endl;
        return 0;
    }

    fstream fichier1(argv[1]); fstream fichier2(argv[2]);

    Polyhedron p1; Polyhedron p2;

    fichier1 >> p1; fichier2 >> p2;

    fichier1.close(); fichier2.close();

    Nef_polyhedron n1(p1); Nef_polyhedron n2(p2);

    n1 -= n2;

    Polyhedron p;

    if(n1.is_simple())
        n1.convert_to_Polyhedron(p);
    else
    {
        cerr << "Le maillage resultant comporte plusieurs parties"
             << endl;
        return 1;
    }

    cout << p;
}
```

Références

[1] http ://www.cgal.org

[2] http ://www.geuz.org/gmsh/doc/texinfo/gmsh.html

[3] Leif Kobbelt, $\sqrt{3}$-Subdivision, Max-Planck Institute for Computer Sciences.

[4] Zhaoyuan Wang, International Journal of Multiphase Flow 35 (2009) 227–246, A coupled level set and volume-of-fluid method for sharp interface simulation of plunging breaking waves, 2008.

[5] Rao V. Garimella, Curvature Estimation for Unstructured Triangulations of Surfaces, *MS B284, T-7, Los Alamos National Laboratory, Los Alamos, NM 87545*

[6] Andrew Kuprat, Volume Conserving Smoothing for Piecewise Linear Curves, Surfaces, and Triple Lines. *Journal of Computational Physics 172, 99–118*, 2001