# Décimation en imagerie médicale 3D

David Coeurjolly

16 mars 1999

Stage effectué au laboratoire ERIC
Université Lumière Lyon II
5 av. Pierre Mendès France
69676 Bron

**Résumé** :Par décimation, on entend processus visant à réduire le nombre de primitive géométrique d'un objet 3D. Cette réduction étant destructrice d'information, le but est donc de trouver un algorithme qui préserve l'aspect visuel de l'objet. Le but de ce rapport est donc dans un premier temps de faire une bibliographie de algorithmes existants en discutant de leurs avantages et inconvénients. Ensuite, après une étude des contraintes sur les images médicales 3D, nous proposons un algorithme adapté à ces images c'est à dire rapide, d'une bonne qualité et non gourmande en mémoire. Une analyse expérimentale est ensuite menée pour valider l'algorithme.

**Mots clés** : décimation, images médicales 3D, spectre de décimation

Responsables de stage :

| | | |
|---|---|---|
| SARRUT David, | Doctorant, | ERIC |
| TOUGNE Laure, | Maître de conférence, | ERIC |

Stage effectué Juillet-Aout 1998

# Table des matières

# Introduction

Avec les développements récents des techniques d'imagerie et de visualisation, la modélisation informatique d'un objet ou d'un phénomène physique devient une approche de plus en plus utilisée en vue de l'analyse d'un problème donné. Cela est particulièrement vrai en imagerie médicale où l'imagerie en 3 dimensions devient un outil incontournable d'aide au diagnostic.

Ainsi, de nombreux appareils d'acquisition d'images sont apparus pour effectuer le passage d'informations du monde réel au monde informatique, la nature de l'information prise en compte variant à chaque méthode. Parmi ces méthodes, nous trouvons deux grandes classes de méthodes d'acquisition en imagerie médicale :

**Imagerie fonctionnelle** ces méthodes d'acquisition sont utilisées pour fournir des informations sur l'activités des organes étudiés (*e.g.* activités neuronales). Dans ce cas, l'information mesurée correspond à des zones d'excitation électromagnétique (Électro-EncéphaloGraphie-EEG, Magneto-EncéphaloGraphie-MEG) ou nucléaire (Résonance Magnétique Nucléaire fonctionnelle-IRMf, tomographie par émission de positon-TEP).

**Imagerie anatomique** dont le but est d'avoir une cartographie basée sur les limites spatiales d'organes ou la discrimination de tissus. Parmi ces méthodes, nous trouvons l'IRM anatomique dont l'information est basée sur la réaction de noyaux d'hydrogène, ce qui nous donne des informations sur la localisation des tissus mous. La tomographie X qui mesure l'aptitude des tissus à absorber les rayons X nous aide par exemple à la localisation des os.

Toutes ces méthodes produisent une image 3D de l'objet analysé, image représentée informatiquement par un pavé discret de $\mathbb{Z}^3$ où un potentiel à chaque noeud caractérise l'information mesurée du milieu.

A partir de ces images, plusieurs traitements sont possibles selon l'utilisation recherchée, nous pouvons soit travailler sur le pavé discret pour, par exemple, faire de la segmentation d'organes précis ou pour dégager des informations sur l'image. Nous pouvons aussi travailler sur une surface 3D que nous obtenons grâce à des algorithmes des Marching-Cubes, cette surface est calculée comme étant la frontière sur le pavé entre les noeuds dont le potentiel est supérieur à un seuil donné et ceux inférieur à celui-ci. Nous obtenons ainsi une surface 3D composée de triangles comme primitives. Cette surface pourra être ensuite visualisé par de nombreuses techniques de rendu (Zbuffer ... ).

Cependant, dans le cadre de traitement ou de visualisation d'une iso-surface, les différents algorithmes de construction de cette iso-surface entraînent la création de nombreuses facettes (exemple du crâne figure 1 : 600 000 facettes). Ainsi, tout traitement de cette surface sera très long car lié au nombre de facettes ou de sommets, c'est particulièrement le cas des algorithmes de visualisation ou de rendu. Pour réduire les durées de traitement, une des méthodes est de réduire le nombre de facettes. Le choix des facettes à supprimer ne pouvant pas se faire au hasard, les algorithmes de décimation apportent donc des méthodes de sélection des éléments de l'objet à supprimer. Par exemple, certaines méthodes reposent sur la suppression de facettes aux endroits de l'objet où la courbure locale est faible ce qui entraîne donc une faible modification de l'aspect de l'objet. Un autre point de vu est basé sur la suppression des petits détails de l'objet comme des petites cavités ou aspérités.

Ainsi, le but de mon stage est de faire une bibliographie des différentes méthodes existantes pour discuter de leurs avantages et inconvénients et de cette bibliographie, je dois dégager un algorithme adapté aux images médicales 3D fortes en contraintes la méthode envisagée. Cet algorithme de décimation devra aussi s'intégrer dans une chaîne de traitement d'images médicales développée au sein du laboratoire ce qui impose encore quelques contraintes sur les structures de données utilisées.

# 1 Problématique et objectifs

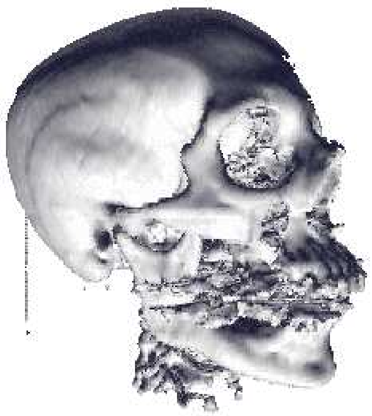

FIG. 1 - *Exemple d'un crâne issu d'une tomographie X, surface extraite par Marching-Cubes.*

## 1.1 Les données du problème

Nous allons donc commencer par l'analyse de la structure d'entrée que nous impose la chaîne de traitements, à savoir une surface issue de l'algorithme des Marching-Cubes. Nous verrons donc les contraintes d'une telle structure ainsi que les objectifs à atteindre.

### 1.1.1 Les Marching-Cubes

Les systèmes d'acquisition comme l'IRM ou la tomographie X, une fois l'étape de reconstruction terminée, produisent une image discrète 3D formée par un ensemble de coupes 2D de l'objet. Cette image 3D est définie sur un pavé de $\mathbb{Z}^3$ où il y a un potentiel sur chaque point $(i, j, k)$ qui caractérise le milieu, calculé par le système d'acquisition, au voisinage de ce point. Nous appellerons ce point un *voxel* .

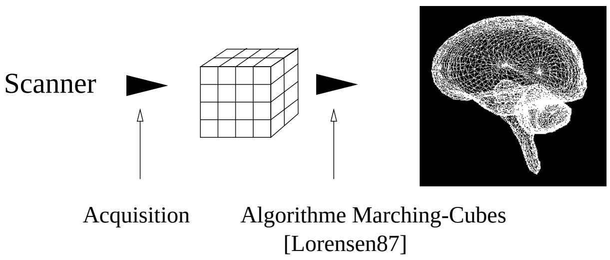


FIG. 2 - *Étapes de la chaîne de traitement en vue d'une visualisation d'une surface 3D.*

Ensuite, à partir de cette structure, nous pouvons calculer une iso-surface en 3D grâce à un algorithme d'extraction de surface triangulée appelé *Marching-Cubes* [LC87] (cf figure 2). Cet algorithme a pour but de construire une surface (sous forme de triangles) sur l'image discrète 3D à partir d'un seuil donné, telle que cette surface partage l'espace en 2 : d'un coté les points dont le potentiel est inférieur au seuil et de l'autre les points dont le potentiel est supérieur au seuil.

De cet algorithme nous pouvons retenir des contraintes fortes sur la structure de la surface :

– Contraintes topologiques : la surface créée est topologiquement correcte (il n'y a pas de triangles qui s'intersectent) et fermée (chaque triangle a exactement 3 voisins).

- Le nombre moyen de triangles incidents à un sommet pour tous les sommets de la triangulation est de 8 [JM97].
- Nous n'avons aucun triangle dégénéré (triangle dont les sommets sont alignés).

**Définition 1** *Soit $t$ un triangle, nous définissons l'aspect du triangle $t$ le rapport entre le plus grand et le plus petit coté du plus petit rectangle circonscrit à $t$.*
*La notion de plus petit triangle est équivalente à celle de rectangle d'aire minimale.*

- L'aspect moyen des triangles est correct dans le sens où l'aspect moyen des triangles est proche de celui d'un triangle équilatéral (aspect moyen de 0.7). Cette mesure est normalisée avec l'aspect d'un triangle isocèle (cf figure 3).

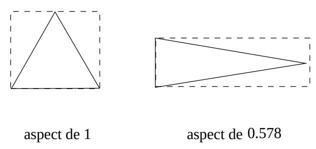


FIG. 3 - *Calcul de l'aspect d'un triangle.*

### 1.1.2 Structure de données

A la sortie des Marching-Cubes, nous obtenons une surface 3D avec des sommets et des facettes liées à ces sommets. Ainsi, nous sauvegardons cette structure sous la forme non-redondante suivante :

- Un tableau de sommets avec pour chaque sommet ses coordonnées réelles (x,y,z) et la normale en ce sommet ($\vec{N}$).
- Un tableau des facettes où chaque case correspond à 3 entiers, indices des sommets dans le premier tableau.

Dans la pratique, il est toujours utile de savoir quels sont les voisins à une face. Pour cela Miguet et al [MNS97] ont proposé un algorithme linéaire qui calcule les voisins de chacun des triangles de l'iso-surface. Nous obtenons donc une nouvelle structure qui tient compte de cette information (cf figure 4).

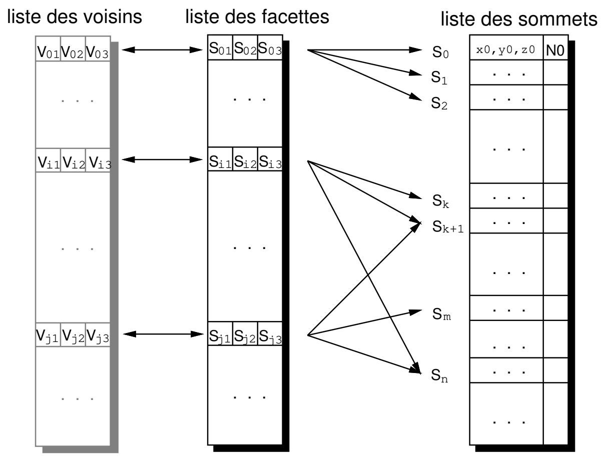


FIG. 4 - *Structure de données.*

## 1.2 Objectifs

Maintenant que nous avons défini la structure d'entrée, nous pouvons définir les objectifs et les contraintes, l'optique principale étant la réduction du nombre de primitives géométriques (triangles dans notre cas).

**Contraintes de place mémoire :** Dans l'exemple du crâne (cf figure 1), notre structure tient déjà sur plus de 10 Mo en mémoire, il est donc nécessaire de limiter les structures annexes dans la décimation. En effet, un simple tableau de marquage des sommets prend déjà plus de 700 Ko en mémoire .

Ainsi, certains algorithmes décrits dans le littérature ne seront pas exploitables avec de telles surfaces 3D à cause de la trop grande taille de tableaux intermédiaires.

**Stabilité et convergence :** D'autres contraintes, plus au niveau de l'implémentation, peuvent apparaître comme la rapidité de la décimation, la création de fonctions réutilisables sous forme de librairies, en vue de l'intégration sur des plates-formes de traitements médicaux comme *aramis* [Sar] développée au laboratoire ERIC par D. Sarrut.

# 2 La décimation

## 2.1 Généralités sur les différents algorithmes

Étant donné l'engouement pour l'optimisation d'objets 3D, de nombreux algorithmes de décimation ont vu le jour en introduisant, presque tout le temps, un nouveau point de vu du problème. Cependant, nous pouvons généraliser à 2 grandes classes d'algorithmes (cf figure 5) :

- Les algorithmes *d'optimisation topologique* dont le principe est de réduire les trous, tunnels ou cavités (et par conséquence les primitives géométriques).
- Les algorithmes *d'optimisation géométrique* dont le but est de réduire le nombre de primitives géométriques telles que les sommets, les triangles ou les arêtes.

Dans le cas d'une optimisation géométrique, nous avons encore de nombreuses méthodes différentes :

- On peut créer un nouvel objet avec moins de primitives et on essaye de l'ajuster pour minimiser l'erreur entre celui-ci et l'objet initial : ce sont les *algorithmes constructifs.*
- On peut chercher à détruire localement une primitive, pour cela nous introduisons un critère de suppression sur celle-ci et ensuite on maintient la structure en triangulant le trou obtenu : ce sont les *algorithmes destructifs.*
- Enfin, on peut essayer de modifier la structure, c'est-à-dire la triangulation même de l'objet, pour réduire le nombre de primitives tout en contrôlant l'erreur avec l'objet initial, ce sont les *algorithmes d'optimisation de la structure.*

### 2.1.1 Optimisation topologique

Dans cette catégorie d'optimisation, le but est de réduire tous les petits détails de l'objet, c'est à dire que nous cherchons à *lisser* la surface de l'objet pour obtenir un objet plus simple, avec moins de primitives. Parmi ces détails, nous avons les cavités qui correspondent à de petites aspéritées de la surface, des trous, et des tunnels. Dans l'optique envisagée, les Marching-Cubes nous garantissent de ne pas avoir les 2 derniers cas. L'analyse spectrale de l'objet nous localisera ces aspérités dans les hautes fréquences, aisni la suppression de ces détails revient donc à appliquer la contrainte suivante :

$$\boxed{\text{Réduire les détails à haute fréquence de l'objet}}$$

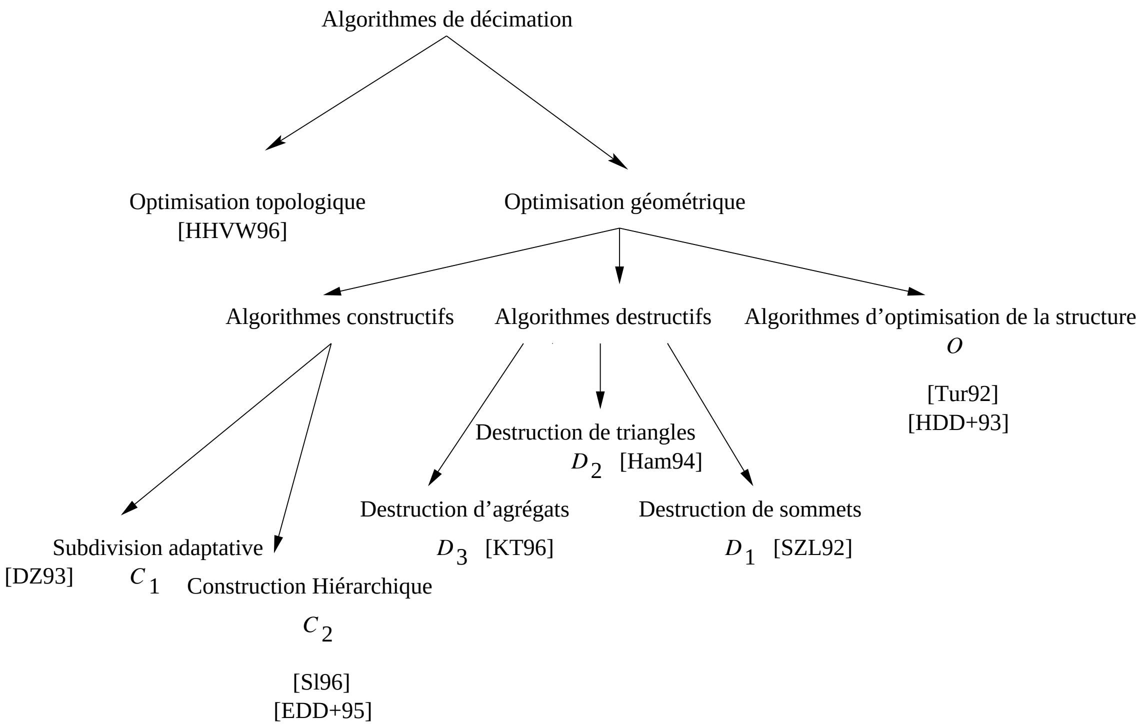


FIG. 5 - *Classification des différents algorithmes.*

En effet, si nous nous plaçons dans le cas d'un traitement de signal 2D, une technique commune de lissage est basée sur l'application d'un filtre passe-bas fréquentiel sur le signal. Par exemple nous calculons le domaine fréquentiel du signal avec la transformation de Fourier, et nous appliquons sur celui-ci un filtre visant à éliminer les hautes fréquences :

$$FFT(f) = FFT(f).H(f)$$

$$H(f) = \begin{cases} 1 & : \quad -k < f < k \\ 0 & : \quad sinon \end{cases}$$

Le passage de cette idée en 3D a été faite par Taosong He et al [HHVW96], qui ont proposé un algorithme lié au principe du Marching-Cubes : il prend en entrée un ensemble de voxels, en appliquant un filtre passe-bas avec différents coefficients $k$, il construit une pyramide d'images 3D (liste de voxcubes ordonnées selon la qualité de l'approximation). Il reconstruit finalement l'iso-surface en appliquant un algorithme de Marching-Cubes classique sur un niveau de la pyramide choisi par l'utilisateur grâce à un paramètre de "qualité". L'iso-surface ainsi obtenue contiendra donc moins de triangles que l'originale.

Pour ne pas avoir une pyramide ayant trop de niveaux, il a aussi développé des fonctions d'interpolation entre 2 niveaux permettant de limiter la construction de la liste. Cependant, le traitement reste long (filtrage du voxcube initial, interpolation si nécessaire, application du Marching-Cube sur le voxcube obtenu) et la suppression de détails n'est peut être pas judicieuse dans l'optique d'une analyse médicale par un médecin .

### 2.1.2 Optimisation géométrique

Lors d'une optimisation géométrique, nous nous intéressons plus à des notions de *courbure* ou de *quasi-planarité* de la surface de l'objet. Nous définissons par *courbure* toute information quantitative sur la distance entre l'ensemble de triangles adjacents considérés et une surface planaire.

Tous les algorithmes présents dans cette classe doivent donc respecter les contraintes suivantes :

| |
|---|
| Réduire le nombre de primitives aux endroits à faible courbure locale<br>Préserver l'aspect des zones à forte courbure |

**Les algorithmes constructifs :**

Par *algorithmes constructifs*, nous entendons algorithmes qui construisent un nouvel objet en relation avec l'objet initial. De cette manière, nous avons 2 approches différentes : l'une de *bas en haut* qui consiste à partir d'un objet de base grossier (un cube par exemple) et à l'affiner pour se rapprocher de l'objet initial, l'autre de *haut en bas* qui consiste à analyser l'objet initial pour ensuite reconstruire des objets moins complexes.

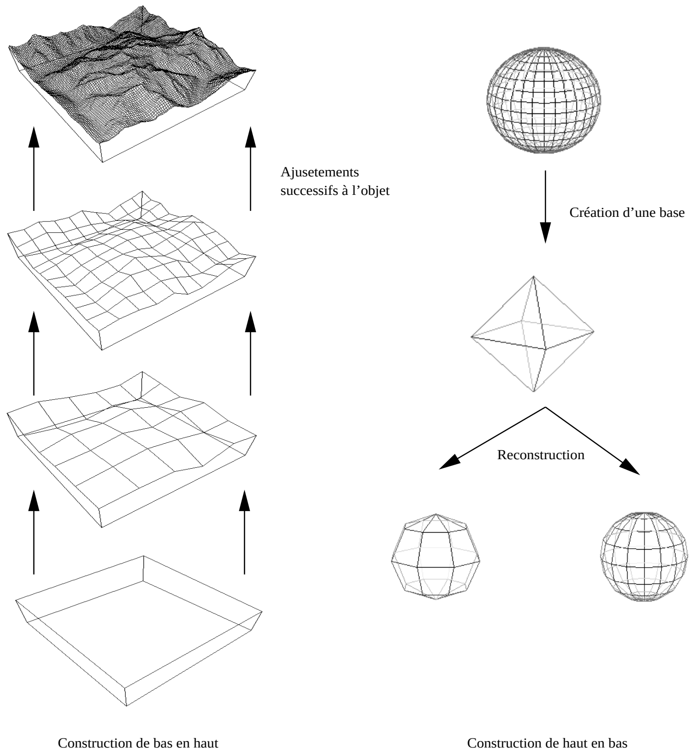


FIG. 6 - *Classes d'algorithmes constructifs.*

Dans une approche de *bas en haut* (cf figure 6) nous avons les algorithmes de subdivision adaptative dont le champ d'application se limite à des MNT (Modèles Numériques de Terrains). Pour ce type d'objets, les algorithmes de subdivision partent d'un plan qui est ensuite "cassé" récursivement en cas de trop grosse différence avec le modèle original, pour arriver enfin à un objet approximant l'original avec moins de primitives. L'emplacement de la rupture ainsi que son mode de division (division en 2 ou 4 polygones) est toujours différent suivant les méthodes. Cependant, DeHaemer et Zyda [DZ93] ont présenté un algorithme de subdivision englobant plusieurs de ces techniques. Ce champ d'application reste uniquement les grilles de potentiels.

Une approche *de haut en bas* consiste à une construction hiérarchique d'objets à partir de l'objet de base, dont le niveau correspond à la qualité de l'approximation choisie. Ainsi, Eck et al [EDD+95] ont proposé une construction basée sur l'analyse en ondelettes de l'objet où le passage d'un niveau à un autre se fait par l'ajustement du niveau précédent avec des coefficients d'ondelettes. Le niveau de base est obtenu par le calcul d'une triangulation de Delaunay sur l'objet initial.

Une seconde méthode, développée par Soucy et Laurendeau [SL96], se base sur une triangulation hiérarchique, c'est-à-dire qu'ils effectuent une optimisation de la triangulation de l'objet en créant différents niveaux de qualité.

**Les algorithmes destructifs:**
Le but de ces algorithmes est de supprimer des éléments géométriques de base de l'objet. Pour cela, nous pouvons généraliser ces algorithmes en 3 étapes :

1. Sélection de l'élément à enlever
2. Suppression de celui-ci
3. Triangulation du trou obtenu pour préserver la topologie de l'objet

Nous pouvons remarquer que la sélection va être un élément prépondérant des algorithmes. Selon la primitive choisie :

**Arêtes** : Nous n'avons pas d'algorithmes basés sur cette primitive car, pour réduire le nombre de sommets ou de triangles d'un objet, il faudrait soit effacer les trois arêtes d'un triangle, soit effacer toutes les arêtes issues d'un même sommet. Ces deux méthodes ne sont pas réalisables car elles nécessiteraient beaucoup d'outils (mémorisation des arêtes à effacer, agrégation) sans avoir un avantage sur les autres algorithmes.

**Sommets** : Schroeder [SZL92] a proposé un algorithme sur la suppression de sommet qui est l'un des plus utilisés car extrêmement simple : on considère un sommet de l'objet, on calcule ensuite l'erreur engendré par la suppression de ce point. Si celle-ci est inférieure à une certaine borne, il l'efface, triangule le trou et recommence. Une analyse en détail de cet algorithme sera faite au paragraphe 3.

**Triangles** : Hamann [Ham94] a proposé un algorithme similaire à celui de Schroeder mais en parcourant les facettes et non les sommets, et en calculant l'erreur d'un triangle sur la courbure des triangles qui l'entourent (courbure obtenues avec les angles internes entre chaque triangle).

**Agrégats** : Un algorithme de cette classe est l'algorithme proposé par Kalvin et Taylor [KT96] basé sur la construction de *super-faces*. Cet algorithme peut être mis en relation avec le calcul des composantes connexes d'une image binaire : on affecte un même label à tous les pixels voisins selon une 4 ou 8-connexité. Dans le cas des super-faces, les pixels correspondent à des triangles et la condition de voisinage de 2 triangles est basée sur un critère de quasi-planarité. Ainsi, un triangle appartient à une super-face (même label au sens des composantes connexes) si l'ajout de ce triangle ne fait pas chuter une contrainte basée sur la quasi-planarité de la super-face. De même, une arête est une *frontière* si elle partage 2 super-faces. Ensuite, Kalvin et Taylor travaillent sur les super-faces obtenues en les lissant puis en les triangulant. Ils obtiennent ainsi une approximation de l'objet initial où les conditions sur l'optimisation géométrique sont préservées.

**Les algorithmes d'optimisation de la structure:**
Dans cette catégorie d'algorithmes, il n'y a pas de construction ou de destruction d'éléments mais une modification locale de la structure de celui-ci. Par exemple, Hoppe et al [HDD+93] proposent d'effectuer localement 3 types d'opérations sur les triangles pour optimiser la structure (cf figure 7). Une fois l'optimisation effectuée, il calcule une fonction d'énergie $\mathcal{E}$ sur le nouvel objet qui décidera si la modification apportée sera gardée ou non. Concrètement, à la $i^{eme}$ itération, si $\mathcal{E}_i < \mathcal{E}_{i+1}$ alors la modification est gardée et on recommence. Ainsi, l'optimisation de l'objet peut se résoudre au calcul du minimum global de la fonction d'énergie.

Dans leur article sur la subdivision adaptative, DeHaemer et Zyda [DZ93] ont présenté une autre façon de voir la décimation : beaucoup d'algorithmes de décimation peuvent se réduire à la recherche d'une combinaison optimale sur un ensemble. Plus simplement, si $\mathcal{S}$ est l'ensemble des sommets de l'objet d'origine et $\mathcal{P}$ l'ensemble des polygones qui forment l'objet à partir de $\mathcal{S}$, alors une simplification de l'objet, pour obtenir un objet à $s$ sommets, peut se réduire en une recherche d'une combinaison $\mathcal{S}'$ de sommets $|\mathcal{S}'| = s$ parmi $\mathcal{S}$ et un ensemble de polygones $\mathcal{P}'$ telle que $\mathcal{P}'$ est *proche* de $\mathcal{P}$.

Ainsi, basé sur cette idée du calcul de $\mathcal{S}' \subset \mathcal{S}$, Turk [Tur92] a proposé un algorithme permettant d'obtenir rapidement $\mathcal{S}'$ et $\mathcal{P}'$. Pour cela, il prend en entrée un nombre de sommets $s$ de l'objet final qu'il distribue au hasard sur la structure (concrètement, il prend au hasard $s$ sommets de $\mathcal{S}$). Ensuite, il applique sur ces points une fonction de répulsion pour les répartir sur l'objet. Puis il détruit tous les autres points en triangulant en même temps à chaque suppression de point, ce qui entraîne un coût assez lourd.

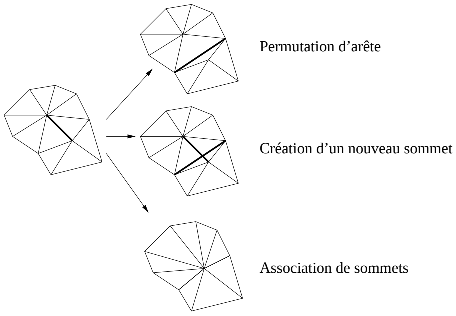


FIG. 7 - *Opérations élémentaires sur les arêtes selon Hoppe.*

## 2.2 Problèmes sous-jacent et critère de qualité

Nous allons maintenant essayer de voir les problèmes communs à tous ces algorithmes.

### 2.2.1 Maintien de la structure

Tout d'abord, tous ces algorithmes maintiennent la structure topologique, c'est-à-dire qu'aucune de ces méthodes ne créent de triangles qui s'intersectent ou de triangles sans exactement 3 voisins.

Cependant, dans le cas des algorithmes destructifs, le problème de la structure est prépondérant. En effet, une fois que l'on a créé un trou, nous devons trianguler celui-ci, pour maintenir la topologie, mais également ajuster les voisins des triangles créés pour ne pas perdre la relation d'adjacence sur les facettes. La triangulation apparaît donc comme un problème important.

### 2.2.2 Critère de qualité

Une fois la scène décimée obtenue, il n'existe pas de méthode de comparaison universelle qui pourrait décider si un algorithme est mieux qu'un autre au sens de la qualité de la décimation. Nous aurions pu essayer de construire une distance entre l'objet décimé et l'objet initial et ainsi quantifier les résultats mais ce n'était pas notre objectif.

Nous jugerons donc nos résultats que sur l'aspect visuel de la scène une fois calculée.

Pour conclure ce chapitre, voici un tableau récapitulatif de quelques algorithmes de décimation avec leurs caractéristiques principales.

| Algorithme | Classe | Param. | Notion erreur globale | Erreur locale | Rapidité | Aspect triangles | Mémoire |
|---|---|---|---|---|---|---|---|
| Schroeder | $\mathcal{D}_1$ | $e_{max}$ | $\times$ | distance plan moyen | ✓ | ✓ | ✓ |
| Haman | $\mathcal{D}_2$ | $e_{max}$ | $\times$ | poids avec angles internes | ✓ | ✓ | ✓ |
| Kalvin | $\mathcal{D}_3$ | $\alpha_{max}$ | $\times$ | angles internes | ?? | $\chi$ | ♣ |
| Turk | $\mathcal{O}$ | $n_{sommets}$ | $\times$ | $\times$ | $\diamondsuit$ | $\times$ | ♠ |
| Hoppe | $\mathcal{O}$ | $ceof_{\mathcal{E}}$ | ✓ | ?? | ?? | $\times$ | ✓ |
| DeHaemer | $\mathcal{C}_1$ | $e_{max}$ | $\times$ | distance avec l'objet initial | ?? | $\times$ | $\heartsuit$ |
| Laurendeau | $\mathcal{C}_2$ | $d_{max}$ | ✓ | ?? | ?? | ?? | ?? |

$\diamondsuit$ : problème dans la reconstruction de l'objet.

✓ : aspect préservé.

$\times$ : non préservé.
$\chi$ : optimisable.
$\clubsuit$ : problème dû au calcul des superfaces (car marquage de toutes les facettes).
$\spadesuit$ : problème car on rajoute $n_{sommets}$ points à l'objet et ensuite on fait disparaitre les autres.
$\heartsuit$ : problème dû à la recopie d'objet en mémoire pour pouvoir comparer deux niveaux.
$e_{max}$ : erreur sans unité définissable de facon empirique.
$\alpha_{max}$ : erreur maximale basée sur les angles internes des triangles qui entourent un triangle donné.
$n_{sommets}$ : nombre de sommets que devra contenir l'objet final.
$ceof_{\mathcal{E}}$ : paramètre d'ajustement de la fonction d'energie. Ce coefficient étant normalisable, nous pouvons le calculer.
$d_{max}$ : distance maximale sur des unités métriques (ex : mm).
?? : caractéristique qui n'existe pas dans l'algorithme initial ou qui n'a pas pu être analysée.

# 3 Un algorithme adapté

## 3.1 Classe d'algorithmes choisie

A cause notamment des contraintes mémoire et du souhait d'avoir un algorithme rapide pour de petites décimations, nous avons choisi de nous intéresser plus en détails aux algorithmes destructifs. En effet, ceux-ci présentent l'avantage de ne travailler que localement sur l'objet, ce qui diminue l'espace mémoire nécessaire à la décimation.

Cependant, ceux-ci ne permettent pas de garantir une erreur globale sur la structure, il faudra donc s'intéresser aussi aux problèmes de propagation d'erreur pour essayer de les diminuer.

Dans cette classe, nous allons nous intéresser plus particulièrement aux algorithmes de destruction de sommets, la destruction d'agrégats comme Kalvin et Russel [KT96] apparaissant elle-aussi coûteuse en mémoire.

## 3.2 Définitions et notations

Nous allons, tout d'abord, définir quelques notations :

**Définition 2** *Nous définissons l'objet comme un triplet $\mathcal{O} = (\mathcal{S}, \mathcal{F}, R)$ avec $\mathcal{S}$ l'ensemble des sommets, $\mathcal{F}$ l'ensemble des faces où chaque face correspond à un triplet de $\mathcal{S}^3$, et $R$ une relation d'adjacence de $\mathcal{S} \to \mathcal{S}$.*

**Définition 3** *Soit $\mathcal{V}$ une fonction bijective de $\mathcal{S} \to \mathcal{P}(\mathcal{F})$ qui a tout sommet de $\mathcal{S}$ associe l'ensemble de facettes qui entoure ce sommet.*

En pratique, cet ensemble est calculé en utilisant la relation d'adjacence à partir d'une face qui contient le sommet en question.

Nous allons ensuite définir l'erreur en un sommet, erreur qui nous servira de critère de suppression de ce sommet.

**Définition 4** *Soit $\mathcal{E}$ une fonction de $\mathcal{S} \to \mathbb{R}$ qui a tout sommet associe une mesure de l'erreur de celui-ci.*

Pour que cette erreur soit en relation avec la courbure locale de l'objet, cette erreur sera calculée à partir de $\mathcal{V}(s)$ ($s$ un sommet).

## 3.3 Analyse de l'algorithme de Schroeder et al

Décrivons maintenant l'algorithme initial de Schroeder et al [SZL92], qui nous sert de base.

Tout d'abord, définissons le critère de Schroeder :

**Définition 5 (Erreur de Schroeder et al)** *$\mathcal{E}$ est définie comme étant la distance entre le sommet considéré est le plan moyen pondéré par les aires de $\mathcal{V}(s)$.*

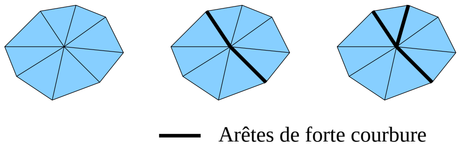


FIG. 8 - *Les différentes classes de sommets selon Schoeder et al: simple, arête intérieure, coin.*

Schroeder et al (cf Annexe C) introduisent dans leur algorithme un classement des sommets en 3 catégories (cf figure 8), par exemple, la classe de sommets *arête* correspond à un sommet dont ses triangles l'entourant possèdent exactement 2 arêtes avec une forte courbure. Ce classement est dû à son calcul du plan moyen de manière à ne pas affecter une erreur faible à un sommet dontla courbure locale est forte (cf figure 9).

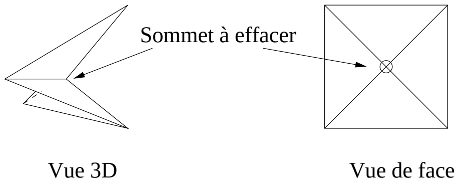


FIG. 9 - *Exemple d'erreur introduite par le plan moyen.*

Ainsi, son calcul d'erreur se fait en 2 temps, le premier pour calculer la distance au plan moyen, le second pour enlever les erreurs dues à ce calcul du plan moyen en retravaillant sur les angles entre chaque facette adjacente de $\mathcal{V}(s)$.

En ce qui concerne l'erreur globale engendrée, cet algorithme, non seulement ne garantit rien, mais de plus il effectue une propagation de l'erreur. En effet, à l'itération $i$, le calcul de l'erreur se fait sur le voisinage du point $s$ de l'itération $(i-1)$ alors que sur l'objet initial, $s$ peut être un point important (au sens de forte courbure locale). Cette propagation entraîne donc une forte erreur globale.

Enfin, la complexité de cet algorithme est en $O(n^2)$ avec $n$ le nombre de sommets, ce qui dans notre optique d'images médicales n'est pas acceptable. Nous ne garderons donc de cet algorithme que les grands principes.

## 3.4 Optimisation de l'algorithme

### 3.4.1 Un nouveau critère de suppression

La première modification que nous apporterons se place sur le calcul de la fonction $\mathcal{E}$. Par ailleurs, Schroeder et al utilisent les angles internes des triangles de $\mathcal{V}(s)$ ainsi que de la distance au plan moyen car la distance seule n'est pas acceptable à cause du cas de la figure 9. En ce qui nous concerne, nous allons nous ramener à un critère plus simple basé uniquement sur les angles internes et non plus sur les angles avec la distance au plan.

Nous définissons notre erreur par :

**Définition 6 (Nouvelle erreur)** *Pour $s \in \mathcal{S}$ :*

$$\mathcal{E}(s) = \alpha Moy(\mathcal{V}(s)) + \beta\sigma(\mathcal{V}(s))$$

*avec $\sigma(\mathcal{V}(s))$ l'écart type des angles entre les normales des faces adjacentes de $\mathcal{V}(s)$ (angles non signés donc de 0 à $\pi$)*
*$Moy(\mathcal{V}(s))$ la moyenne de ces angles et $\alpha$, $\beta$ des coefficients.*

Ainsi, une erreur forte en un sommet caractérise :

– Soit une forte distance au plan moyen car la moyenne des angles est forte.

– Soit une forte perturbation mais distance au plan moyen faible (écart type élevé), ce qui nous garantit de ne pas tomber dans le cas de la figure 9.

– Soit les deux.

L'erreur sera faible quand la moyenne et la variance seront faibles, ce qui caractérise une surface quasi-planaire.

Dans la pratique, les deux coefficients $\alpha$ et $\beta$ sont tous les deux égaux à 1 (ainsi on donne autant d'importance aux deux mesures). De plus, avec ces coefficients notre erreur est bornable (ce qui n'est pas possible chez Schroeder) sa borne maximale étant de $\pi + \frac{\pi}{2}$ et sa borne minimale de 0 (cas planaire).

### 3.4.2 Structure de l'algorithme

A cause des structures manipulées, il serait souhaitable de ne faire qu'un seul parcours des sommets, en n'effaçant que les sommets dont l'erreur est en dessous d'un certain seuil, erreur calculée sur l'objet initial. Ainsi, notre algorithme se divise en 2 grandes étapes :

1. Parcours des sommets et marquage des sommets à effacer.

2. Parcours des sommets marqués et suppression de ceux-ci.

De cette manière, nous garantissons d'enlever tous les sommets dont l'erreur est inférieure à un certain seuil et uniquement ceux-ci.

De plus, nous ne parcourons au maximum que 2 fois chaque sommet (on ne le parcourt qu'une seule fois si le sommet n'est pas à effacer) .

### 3.4.3 Analyse du spectre de décimation

Il nous reste cependant le problème de l'approche de l'erreur passée en paramètre, pour cela, nous introduisons ce que l'on appellera *spectre de décimation* :

> **Définition 7 (Spectre de décimation)** *Nous appellerons spectre de décimation l'histogramme généré par la discrétisation de la fonction $\mathcal{E}(s)$ pour $s \in \mathcal{S}$*

Grâce à ce spectre, nous pouvons calculer l'erreur qu'il faut donner à l'algorithme pour obtenir un certain pourcentage de suppression. L'utilisateur n'a donc plus qu'à donner un pourcentage plutôt qu'une erreur réelle non-significative. Le grain de la discrétisation est laissé aussi en paramètre à l'utilisateur qui pourra affiner le calcul de l'erreur s'il le désire.

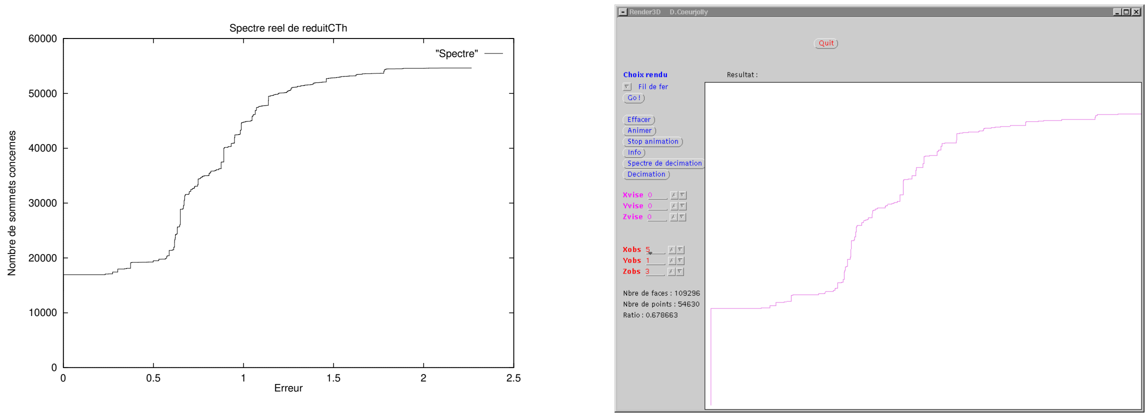


FIG. 10 - *Spectre réel et spectre discret d'un crâne*

### 3.4.4 Triangulation 3D

Maintenant que nous savons sélectionner un point à effacer, il faut savoir trianguler le trou, ou plutôt il faut pouvoir trianguler un polygone 3D formé par tous les sommets des triangles de $\mathcal{V}(s)$ sans $s$. En général, la triangulation 3D d'un champ de points est un problème très étudié par sa complexité mais dans notre cas, nous avons une information forte qui est l'existence d'un point central $(s)$ qui représente "l'intérieur" du polygone. Nous sommes dans le cas d'une triangulation d'un polygone étoilé (cf figure 11).

Ainsi, il faut maintenant choisir une méthode de triangulation 2D pour pouvoir ensuite, soit l'extrapoler en 3D, soit trianguler le projeté de tous les points sur le plan moyen, et ensuite ramener le tout en 3D. Cependant, nous avons vu que le calcul du plan moyen n'est pas fiable (cf paragraphe 3.3) et que le coût des différentes projections risque d'être pénalisant. Donc la dernière méthode ne parait pas acceptable.

Nous avons donc choisi de partir d'une triangulation 2D décrite par Kong et al [KET90] qui est basé sur la notion de coupage "d'oreilles" (cf Annexe A). Cet algorithme a l'avantage d'être en $O(n(k+1))$ avec $n$ le nombre de sommets du polygone et $k$ le nombre de points concaves de ce polygone (dans le cas d'une surface issue du Marching-Cubes, $k$ sera faible voire nul). De plus, le nombre de triangles créé est exactement $(n-3)$.

Il nous faut maintenant redéfinir l'outil principal de cet algorithme pour pouvoir passer en 3D :

**Définition 8 (Point concave 3D)** *Nous appellerons point concave, un point $p$ d'un polygone 3D de centre $s$, contenu dans le triangle formé par $s$, le prédécesseur de $p$ dans la liste des sommets du polygone, et son successeur.*

Nous avons défini un point concave de la sorte car le calcul d'un angle entraîne des appels à des fonctions trigonométriques très coûteux en temps de calcul. Notre calcul de projection se fait, lui, très rapidement en regardant d'abord si le point est contenu dans le triangle puis en calculant la distance de ce point au plan (cf Annexe B).

De plus, dans le cas d'une triangulation d'un polygone 3D plan, notre définition de point concave rejoint celle en 2D car si le point $p$ se projette dans le triangle $pred(p)$, $centre$ et $succ(p)$, cela signifie que l'angle externe $(\widehat{pred(p), p, succ(p)})$ est supérieur à $\pi$.

Ensuite, le reste de l'algorithme 3D est strictement identique à celui de Kong et al, mis à par le fait que nous avons remplacé la structure de donnée en liste, trop coûteuse, par une structure en tableau de points en utilisant des compteurs et une pile pour mimer le chaînage de la liste. De plus, l'ajustement des voisins et la création des nouvelles facettes se fait juste après la détection de l'oreille.

Ainsi, nous passons en paramètre à notre triangulation 4 tableaux (cf figure 11) :

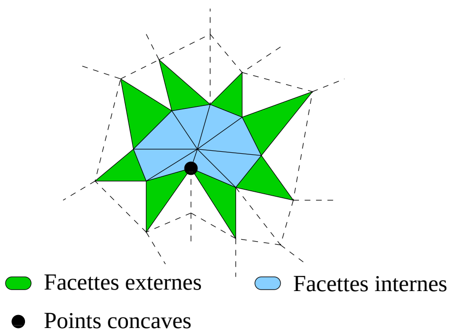


FIG. 11 - *Exemple d'un polygone 3D à trianguler.*

1. Un tableau des sommets qui forment le polygone.

2. Un tableau des facettes internes qui seront à supprimer. Ces facettes nous serviront de cases vides dans le tableau pour pouvoir y insérer les triangles que l'on va créer. Ainsi la triangulation se fait localement dans la structure même.

3. Un tableau des points concaves, calculé lors de la création de $\mathcal{V}(s)$, et en même temps que la création du tableau des sommets.
4. Un tableau des faces externes pour pouvoir ajuster correctement les voisins entourant le polygone. Ce problème de l'ajustement de voisins à l'intérieur comme à l'extérieur de la triangulation est primordial pour maintenir la structure de la surface.

Cependant, la version 3D de cet algorithme n'est pas totalement stable, en effet, il existe des problèmes dans certaines configuration particulière (lorsqu'une face voisine externe a son troisième sommet qui appartient au polygone). Dans ce cas de figure, nous avons choisi de refuser la triangulation et donc de rejeter le point en ne le supprimant pas. L'amélioration de la triangulation dans ces cas particuliers est une des perspectives envisagées.

### 3.4.5 Optimisation par basculement d'arêtes

Lors de notre triangulation, aucune attention n'est portée sur l'aspect des triangles générés. Pourtant, d'après le principe du coupage d'oreille, l'aspect des triangles est assez mauvais, surtout dans le cas d'un polygone convexe (cf figure 12).

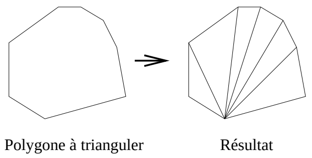


FIG. 12 - *Exemple de triangulation d'un polygone convexe.*

Aussi, une optimisation est nécessaire. Pour cela nous allons effectuer un "basculement d'arête" sur les triangles que l'on crée (cf figure 13).

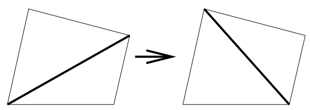

FIG. 13 - *Opération de basculement d'arête.*

En formalisant le problème nous définissons :

**Définition 9 (Fonction d'aspect)** *Soit $\mathcal{A}$ une fonction de $\mathcal{F} \to \mathbb{R}$ qui à tout triangle $t$ fait correspondre la distance entre l'aspect du triangle et l'aspect idéal :*

$$\mathcal{A}(t) = |1-\texttt{aspect(t)}|$$

*la fonction* `aspect(t)` *renvoie l'aspect du triangle $t$ défini dans la définition 1.*

**Définition 10 (Fonction d'énergie)** *Soit $\Phi$ une fonction de $\mathcal{P}(\mathcal{F}) \to \mathbb{R}$ qui à tout ensemble connexe de triangle $C$ associe :*

$$\Phi(C) = \sum_{t \in C} A(t)$$

Soit $\mathcal{T}$ l'ensemble des triangles créés par la triangulation du trou. Nous appellerons `Swap(t1,t2)` (avec `t1` et `t2` des numéros de facettes) la fonction qui permute l'arête commune à t1 et t2 par effet de bord sur la structure.

L'optimisation optimale serait alors :

**Optimise_Triangles**($\mathcal{T}$)
Soit $min = \Phi(\mathcal{T})$
  Pour tout $t \in \mathcal{T}$
    Pour tout $t' \in \mathcal{T}$ et $t' \neq t$
      $\mathcal{T}'$ =`Swap(t,t')`
      Si $(\Phi(\mathcal{T}') < min)$ Alors
        **Optimise_Triangles**($(T')$)
    FinPour
  FinPour
Fin(Optimise_Triangle)

Bien entendu, cette solution est exponentielle et n'est donc pas raisonnable. L'optimisation que nous avons implémenté correspond en fait à une version gloutonne de l'algorithme précédent qui n'est pas optimale mais a de bon résultats (cf figure 14).

FIG. 14 - *Rendu fil de fer d'un cube avec et sans l'optimisation des triangles.*

La difficulté à approcher une solution optimale vient principalement du fait que la structure n'est pas constante au cours du traitement.

### 3.4.6 Importance de l'ordre de parcours

Nous avons aussi remarqué l'importance de l'ordre de parcours et de suppression des sommets. En effet, si nous parcourons le tableau dans l'ordre des indices, à cause de la construction de la surface par les Marching-Cubes, nous aurons tendance à fortement effacer les sommets de manière locale avant de se déplacer sur la surface. Cela est principalement dû au fait que lors de la création de la surface les indices sont créés à la volée, donc des sommets d'indices proches impliquent qu'ils sont proches sur la surface.

On peut aussi se demander si un parcours par ordre croissant des erreurs n'est pas plus judicieux. Pour cela, nous avons créé une structure de parcours qui contient tous les sommets à effacer et le mode de remplissage de ce tableau se fait par un paramètre laissé à l'utilisateur avec 3 possibilités :

1. Parcours selon les indices, donc forte localité.
2. Parcours par ordre croissant des erreurs.
3. Parcours par ordre décroissant des erreurs.

Ainsi, l'expérimentation nous a montré que le mode de parcours 1 est certes rapide (aucun tri à faire) mais le rendu visuel est beaucoup moins bon que pour le parcours 2 par exemple (cf figure 15). Dans nos tests finaux, nous n'avons utilisé que le mode 1 et le mode 2.

FIG. 15 - *Rendu d'un tore décimé avec un parcours normal (gauche) et un parcours trié (droite).*

### 3.4.7 L'algorithme final

Nous avons maintenant tous les éléments pour construire l'architecture de notre algorithme :

1. Parcours des sommets, calcul de l'erreur pour chaque sommet.
2. Calcul du spectre à partir du tableau des erreurs.
3. Calcul de l'erreur en fonction d'un pourcentage de suppression.
4. Calcul du tableau de parcours selon le mode choisi.
5. Suppression des sommets dans l'ordre du tableau du parcours :
   (a) On calcul les tableaux nécessaires à la triangulation ($\mathcal{V}(s)$).
   (b) On triangule le polygone.
   (c) On optimise la triangulation (optionnel).
   (d) On supprime le sommet dans la structure globale.
6. On effectue un petit "ramasse-miettes" pour enlever les cases vides dans le tableau des sommets et dans le tableau des facettes.

## 3.5 Résultats

Nous allons pourvoir maintenant tester les performances de notre algorithme et enfin voir s'il répond aux objectifs : rapidité et qualité.

### 3.5.1 Analyse de la mémoire

Au niveau de la mémoire, tout se résume dans la figure 16. Nous pouvons remarquer que le coût en place mémoire d'une décimation reste raisonnable grâce à une gestion dynamique des données intermédiaires.

Par exemple, dans le cas du crâne `reduitCth`, la structure initiale (sommets, normales, facettes et voisins) prend 4.356 Mo et au maximum de la décimation 5.081 Mo, soit une augmentation de seulement 14.3 % .

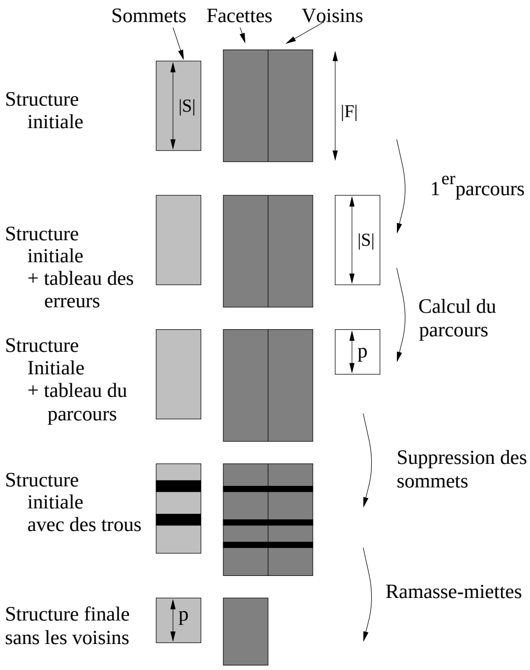


FIG. 16 - *Objets en mémoire durant le traitement avec $|\mathcal{S}|$ le nombre de sommets, $|\mathcal{F}|$ le nombre de facettes et $p$ le nombre de sommets à effacer.*

### 3.5.2 Analyse de la complexité

En complexité, nous avons (on note $p$ le pourcentage de suppression) :

| Désignation | coût |
|---|---|
| Calcul du tableau d'erreur | $O(\|\mathcal{S}\|)$ |
| Calcul du spectre | $O(\|\mathcal{S}\|)$ |
| Calcul de l'erreur en fonction de $p$ | $O(nombre\ de\ discretisations)$ |
| Calcul du mode de parcours | $O(\|\mathcal{S}\|)$ si mode normal |
| | $O(\|\mathcal{S}\|\log(\|\mathcal{S}\|))$ si mode trié |
| Suppression des points | $O(p)$ |
| Ramasse-miettes | $O(\|\mathcal{S}\| + \|\mathcal{F}\|)$ |

De plus, notre triangulation se fait en $O(n(k+2))$ avec $k$ le nombre de points concaves d'un polygone à n sommets (on a un parcours supplémentaire pour détecter les sommets à rejeter, grâce notamment, à un marquage des sommets du polygone).

### 3.5.3 Implémentation

L'implémentation de cet algorithme s'est faite avec la création de 6 librairies C :

**VoisinIO** qui comprend toutes les fonctions d'entrée-sortie avec notre structure ainsi que de l'interfaçage des fonctions pour le calcul des voisins.

**List** : une librairie pour les opérations sur les listes.

**Disc** : librairie de création et d'utilisation d'un tableau discret.

**Triangulation** : le module de triangulation d'un polygone 3D.

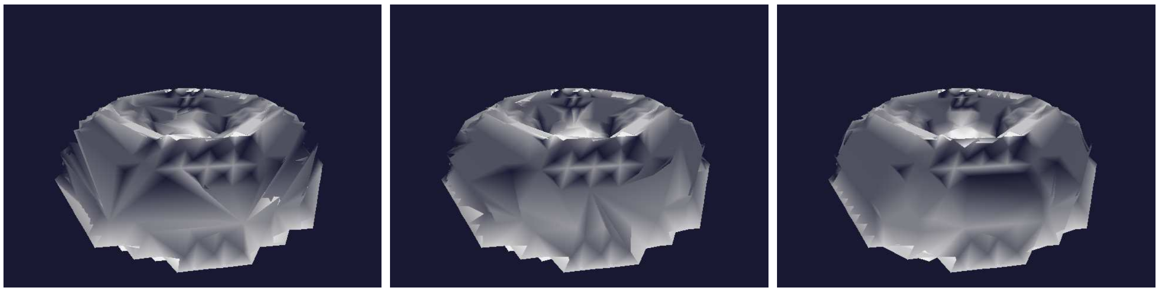

FIG. 17 - *Le tore : 25% normal (gauche), 25% trié (centre) et 25% trié et optimisé (droite).*

**Bitmap** : librairie pour la création d'un tableau de bits pour marquer les sommets (création, mise à 1 d'un bit ... )

**EdgeSwapping** : librairie qui contient les fonctions d'optimisation de la triangulation.

**Render** : librairie générale qui contient les fonctions de décimation, de calcul d'erreur ...

Toutes ces librairies sont utilisées par 2 exécutables, le premier pour une décimation en ligne, avec le passage d'arguments, le second avec une interface sous X11 et OpenWindows qui nous permet en plus de visualiser le spectre et d'avoir un rendu fil de fer de l'objet.

### 3.5.4 Quelques résultats en image

Toutes les durées sont données en secondes.

| Fichier | % de suppression | Mode parcours | Optimisation | Durée décimation | Figure |
|---|---|---|---|---|---|
| `tore` | 50% | normal | × | 0.47 | 17 gauche |
| | 50% | trié | × | 0.48 | 17 centre |
| | 50% | trié | ✓ | 0.77 | 17 droite |
| `reduitCTh` | 25% | normal | × | 11.77 | 18 gauche |
| | 25% | trié | ✓ | 18.06 | 18 centre |
| | 50% | trié | ✓ | 20.89 | 18 droite |

TAB. 1 - *Résultats des décimations.*

| Nom | Nombre de sommets | Nombre de facettes | Figure |
|---|---|---|---|
| `tore` | 2096 | 4192 | 19 gauche |
| `reduitCTh` | 63318 | 127008 | 19 droite |

TAB. 2 - *Données sur les fichiers.*

Pour chaque scène, tous les tests ont été exécutés avec les mêmes paramètres. La figure 20 représente le détail des décimations du tore.

# 4 Conclusion et Perspectives

## 4.1 Conclusion

En conclusion, notre algorithme semble avoir des propriétés intéressantes vis à vis de la mémoire et de la complexité. Cependant, le mode de triangulation que nous avons choisi, aussi simple qu'il y parait,

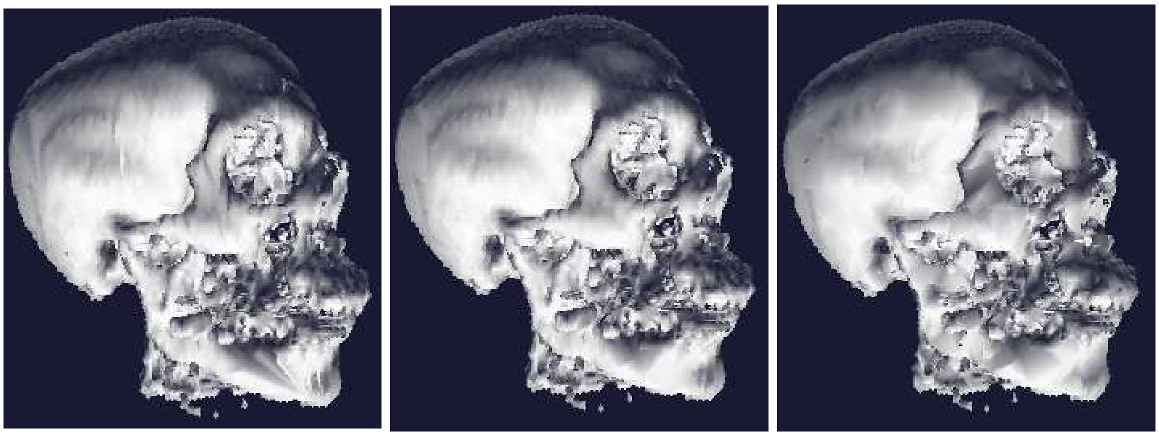

FIG. 18 - *Le crâne : 25% normal (gauche), 25% tiré et optimisé (centre) et 50% trié et optimisé (droite).*

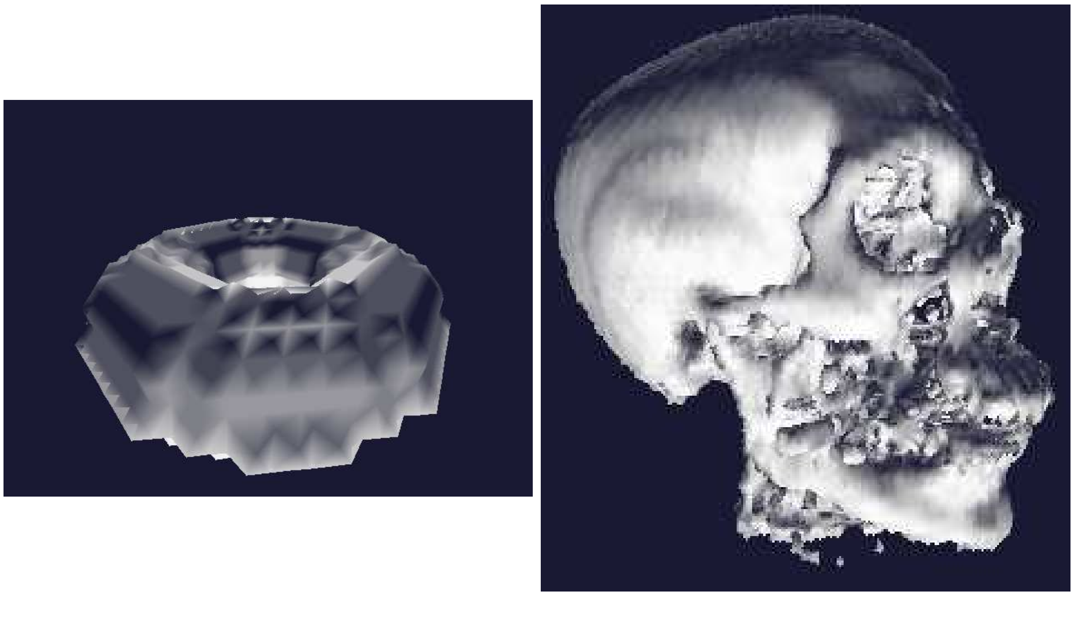

FIG. 19 - *Les versions initiales : tore (gauche) et crâne (droite).*

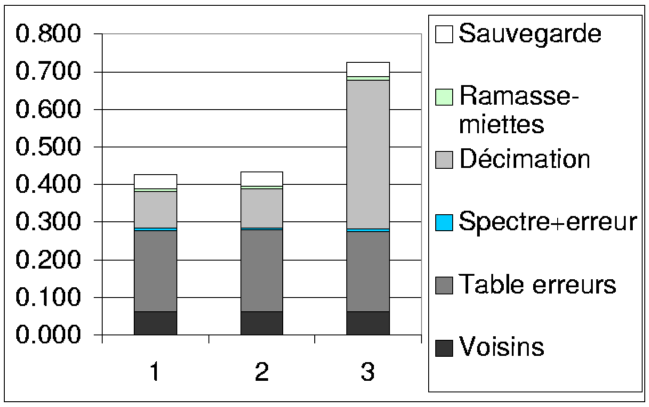


FIG. 20 - *Détail de la durée en secondes des traitements pour le tore décimé à 50% en mode normal (1), trié (2) et trié et optimisé (3).*

nous a posé pas mal de problèmes pour stabiliser son comportement sur des polygones un peu complexes (segments colinéaires, points alignés) pour qu'il n'y ait pas de création de facettes dégénérées par exemple. Ce problème de la triangulation n'est malheureusement pas entièrement résolu ce qui entraîne des instabilités du programme lors de décimation trop importante.

La structure même de l'algorithme paraissant adaptée, la stabilisation de la triangulation entraînerait, selon nous, de bon résultats en rapidité et en qualité. Nous avons pu en effet constater de bon résultats sur des décimation allant jusqu'à 50 %.

## 4.2 Perspectives

Tout d'abord, des améliorations des problèmes de notre algorithme :

**Optimisation des arêtes** : Il serait judicieux d'améliorer notre version gloutonne de cette optimisation qui n'est pas optimale, ce problème de minimisation de fonction peut jouer un rôle important dans tout algorithme de décimation.

**Triangulation** : Il faudrait, soit trouver une nouvelle méthode de triangulation plus adaptée aux polygones 3D, soit stabiliser la version actuelle. Ce problème reste le plus important mais les résultats déjà obtenus semble trés prometteurs (rapidité, mémoire, qualité).

**Calcul de l'erreur en fonction du spectre** : Là aussi on pourrait envisager une amélioration de ce calcul en calculant, par exemple, le polynôme interpolateur du spectre pour affiner l'approximation de l'erreur.

Par contre, nous pouvons aussi envisager d'autres approches de la décimation, par exemple, dans le cadre d'un rendu précis, on pourrait vouloir une erreur faible sur les sommets visibles ou proches de l'observateur et une erreur plus forte sur des sommets cachés ou lointain. Nous aurons, ainsi, une erreur variable sur l'objet même. De plus, cette idée de distance avec l'observateur peut-être prépondérante, par exemple dans le cas d'un rendu d'un crâne vu de très loin, il ne sera pas nécessaire alors de garantir une erreur faible car les algorithmes de rendu (zbuffer par exemple) parcourent toutes les facettes même si celles-ci sont petites.

Une approche de cette idée pourrait être une pyramide de décimations, avec des fonctions pour passer rapidement d'un niveau à l'autre, et le niveau, dans la pyramide, sélectionné serait en fonction de la distance à l'observateur.

# A La triangulation par balayage de Graham

Nous allons étudier plus en détail la version 2D de l'algorithme de triangulation par coupage d'oreille. Quelques définitions :

**Définition 11 (Sommet convexe d'un polygone)** *Nous appellerons un point convexe p d'un polygone tout point tel que l'angle externe entre les segments* $(pred(p), p)$ *et* $(p, succ(p))$ *est inférieur à* $\pi$.

**Définition 12 (Sommet concave d'un polygone)** *Nous appellerons un point convexe p d'un polygone tout point tel que l'angle externe entre les segments* $(pred(p), p)$ *et* $(p, succ(p))$ *est supérieur à* $\pi$.

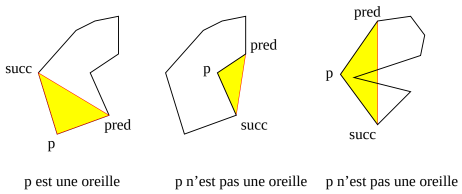


FIG. 21 - *Définition d'une oreille.*

Le but des algorithmes de triangulation par coupage d'oreille est de détecter une oreille, de la couper, puis de recommencer jusqu'à arriver à un polygone à 3 sommets. L'arrêt de ces algorithmes est basé sur le théorème de 2 oreilles :

**Théorème 1 (2 oreilles)** *Excepté pour les triangles, tout polygone convexe a au moins deux oreilles disjointes.*

Ce théorème a été prouvé par Meisters [Mei75] par induction sur le nombre de sommets du polygone. Ainsi, le balayage de Graham est une méthode de détection d'oreille qui repose sur les faits suivant :

1. tout sommet concave n'est pas une oreille.
2. Un sommet convexe est une oreille si et seulement s'il n'existe pas de points concaves à l'intérieur du triangle formé par le point étudié, son prédécesseur dans la liste des sommets et son successeur.

Nous pouvons tout de suite remarquer que la seconde règle sera à l'origine de la complexité en $O(k)$ avec k le nombre de points concaves. Nous avons donc la fonction de détection d'une oreille :

**IsAnEars**($s$,$P$,$C$)
  Si $C$ est vide rendre `Vrai`
  Sinon
    Si $s$ est concave rendre `Faux`
    Sinon
      Si aucun des points concaves ne sont dans le triangle $(pred(p), p, succ(p))$ rendre `Vrai`
      Sinon rendre `Faux`
**Fin(IsAnEar)**

avec $s$ un sommet du polygone $P$ et $C$ la liste des sommets concaves de $P$

Nous avons donc la procédure de triangulation à partir de cette fonction :

**Triangule**($P$,$C$)
$s_i \leftarrow s_2$

Tant que $s_i \neq s_0$
  Si IsAnEar(pred($s_i$),$P$,$C$)
    On crée le triangle $(s_i, pred(s_i), pred(pred(s_i)))$
    On efface $pred(s_i)$ de $P$
    Si $s_i \in C$ et $s_i$ devenu concave
      On supprime $s_i$ de $C$
    Si $pred(s_i) \in C$ et $pred(s_i)$ devenu concave
      On supprime $pred(s_i)$ de $C$
    Si $pred(s_i) = s_0$
      $s_i \leftarrow succ(s_i)$
  Sinon
    $s_i \leftarrow succ(s_i)$
FinTantQue
**Fin(Triangule)**

# B Détection rapide de la projection d'un point sur un triangle

Nous allons décrire une méthode rapide pour savoir si un point 3D se projette sur un triangle donné. Tout d'abord, quelques définitions :

**Définition 13 (Volume signé)** *Nous appellerons volume signé $V(\vec{u}, \vec{v}, \vec{w})$ le déterminant de la matrice formée par $(\vec{u}, \vec{v}, \vec{w})$. Ainsi :*

$$V(\vec{u}, \vec{v}, \vec{w}) = \begin{vmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{vmatrix}$$

Soient A,B,C les points du triangle et M le point à tester. On note aussi $\vec{n}$ le vecteur normal à ce triangle. Pour que M se projette dans le triangle (ABC), il faut et il suffit que M soit dans le prisme ayant pour base le triangle (cf figure 22). Pour cela, nous allons tester la position de M par rapport aux 3 plans $(\vec{AB}, \vec{n})$, $(\vec{AC}, \vec{n})$ et $(\vec{BC}, \vec{n})$.

Donc pour le premier plan, il faut que M soit du même côté de ce plan que C, de même côté que B pour le second et que A pour le dernier.

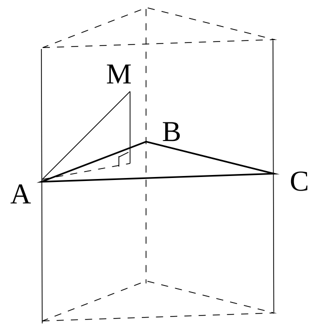


FIG. 22 - *Prisme de projection.*

Pour effectuer ce test, nous définissons :

**Définition 14 (Même coté que)** *Deux points M et N sont du même coté par rapport au plan $(\vec{AB}, \vec{n})$ si et seulement si :*

$$V(\vec{AB}, \vec{n}, \vec{AM}) \times V(\vec{AB}, \vec{n}, \vec{AN}) > 0$$

Ainsi, M se projette sur le triangle ABC si et seulement si :

$$V(\vec{AB}, \vec{n}, \vec{AM}) \times V(\vec{AB}, \vec{n}, \vec{AC}) > 0$$

Et

$$V(\vec{AC}, \vec{n}, \vec{AM}) \times V(\vec{AC}, \vec{n}, \vec{AB}) > 0$$

Et

$$V(\vec{BC}, \vec{n}, \vec{BM}) \times V(\vec{BC}, \vec{n}, \vec{AB}) > 0$$

En décomposant le calcul du déterminant, nous pouvons garder de nombreux résultats intermédiaires pour réduire de nombre d'opérations élémentaires, ce qui nous donne une méthode de test très rapide.

Dans le cadre de la triangulation, nous testerons d'abord si la distance du point au triangle est nulle avant de tester les sens ce qui nous donnera un test pour décider si un point 3D est contenu dans un triangle.

# C Algorithme de Schroeder et al

Voici l'algorithme initial de Schroeder et al :

```
Tant qu'il n'y a pas eu un passage sans modification
  Pour tout s ∈ S
    Soit P le plan moyen des triangles de V(s)
    On calcule ε = E(s)
    Si ε < ε_max
      alors
        Si s est de la classe simple
          alors
            Calculer la triangulation du trou formé par la suppression de s
            Si la triangulation est valide
              alors
                On efface s de la structure de donnée et on ajoute les arêtes de la triangulation
              sinon
                On n'efface pas s
          sinon
            Si V(s) ont des aires petites
              alors
                Calculer la triangulation du trou formé par la suppression de s
                Si la triangulation est valide
                  alors
                    On efface s de la structure de donnée et on rajoute le résultat de la triangulation
                  sinon
                    On n'efface pas s
              sinon
                Ne rien faire
      sinon
        Ne rien faire
  FinPour
FinTantQue
```

# Références

[DZ93] H. J. DeHaemer et M. J. Zyda. « Simplification of Objects Rendered by Polygonal Approximations ». *Computers & Graphics*, 15(2):175–184, 1993.

[EDD+95] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, et Werner Stuetzle. « Multiresolution Analysis of Arbitrary Meshes ». Dans Robert Cook, éditeur, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 173–182. ACM SIGGRAPH, Addison Wesley, août 1995. held in Los Angeles, California, 06-11 August 1995.

[Ham94] Bernd Hamann. « A data reduction scheme for triangulated surfaces ». *Computer-Aided Geometric Design*, 11:197–214, 1994.

[HDD+93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, et Werner Stuetzle. « Mesh Optimization ». Dans James T. Kajiya, éditeur, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 19–26, août 1993.

[HHVW96] Taosong He, Lichan Hong, Amitabh Varshney, et Sidney W. Wang. « Controlled Topology Simplification ». *IEEE Transactions on Visualization and Computer Graphics*, 2(2):171–184, juin 1996.

[JM97] Nicod Jean-Marc. « *Extraction de surfaces en imagerie médicale: approches parallèles.* ». PhD thesis, École Normale Superieur de Lyon, 1997.

[KET90] X. Kong, H. Everett, et G.T. Toussaint. « The Graham scan triangulates simple polygons ». *Pattern Recognition Letters*, 1990.

[KT96] Alan D. Kalvin et Russell H. Taylor. « Superfaces: Polygonal Mesh Simplification with Bounded Error ». *IEEE Computer Graphics and Applications*, 16(3):64–77, mai 1996.

[LC87] W. E. Lorensen et H. E. Cline. « Marching Cubes: a High Resolution 3D Surface Construction Algorithm ». Dans M. C. Stone, éditeur, *SIGGRAPH '87 Conference Proceedings (Anaheim, CA, July 27–31, 1987)*, pages 163–170. Computer Graphics, Volume 21, Number 4, juillet 1987.

[Mei75] G.H Meisters. « Polygons have ears ». *American Mathematical Monthly*, 1975.

[MNS97] S. Miguet, J. M. Nicod, et D. Sarrut. « A Linear Algorithm for Constructing the Polygon Adjacency Relation in Iso-surfaces of 3D Images ». *Lecture Notes in Computer Science*, 1347:125–136, 1997.

[Sar] David Sarrut. « ARAMIS : an "on line" parallel platform for medical imaging ». Dans *International Conference on Parallel and Distributed Processing Techniques and Applications.*

[SL96] Marc Soucy et Denis Laurendeau. « Multiresolution Surface Modeling Based on Hierarchical Triangulation ». *Computer Vision and Image Understanding: CVIU*, 63(1):1–14, janvier 1996.

[SZL92] William J. Schroeder, Jonathan A. Zarge, et William E. Lorensen. « Decimation of triangle meshes ». *Computer Graphics*, 26(2):65–70, juillet 1992.

[Tur92] Greg Turk. « Re-tiling polygonal surfaces ». volume 26, pages 55–64, juillet 1992.