Implantation de structures de données compactes pour les triangulations

Abdelkrim Mebarki

UNIVERSITE DE NICE-SOPHIA ANTIPOLIS

ECOLE DOCTORALE STIC

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

T H E S E

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice-Sophia Antipolis

Mention : Informatique

présentée et soutenue par

Abdelkrim MEBARKI

Implantation de structures de données compactes pour les triangulations

Thèse dirigée par *Olivier DEVILLERS*

soutenue le *15 Avril 2008*

Rapporteurs :

M. Jean-Michel MOREAU	professeur
M. Gilles SCHAEFFER	directeur de Recherche CNRS

Jury :

Jean-Michel MOREAU	professeur	rapporteur
Jérôme GALTIER	ingénieur Orange Labs	examinateur
André LIEUTIER	ingénieur Dassault Systèmes	examinateur
Olivier DEVILLERS	directeur de recherche INRIA	directeur de thèse
Luca CASTELLI ALEARDI	docteur	invité

Remerciements

Je tiens à remercier Olivier Devillers, mon directeur de thèse, pour avoir dirigé ce travail. Pour m'avoir accueilli au sein du projet GEOMETRICA durant mon DEA, et pendant ma thèse. Pour son soutien durant toutes mes études en France. Pour sa disponibilité, son aide et son suivi pédagogique et administratif durant ma formation.

Je remercie Jean-Michel Moreau et Gilles Schaeffer pour l'attention qu'ils ont porté à mon travail, et pour avoir accepté d'être rapporteurs de ma thèse.

Je remercie Jérôme Galtier et André Lieutier qui ont bien voulu accepter de participer au jury de ma thèse.

Je remercie Luca Castelli Aleardi pour son aide précieuse durant ma thèse, et pour avoir accepté d'assister à ma soutenance.

Je tiens à exprimer mes vifs remerciements à Agnès Bessière pour sa générosité, et son soutien durant mon séjour à GEOMETRICA.

Je remercie tous les membres du projet GEOMETRICA, permanents, doctorants, ingénieurs et stagiaires, avec qui j'ai passé trois ans. Pour leurs soutiens, Jean-Daniel, Frédéric, David, Steve, Monique, Mariette, Benjamin, Samuel, Manuel, Sébastien, Pedro, Quentin, Pooran, Trung, Nader, Jane, Camille, Andreas, Laurent S., Laurent R., Flavien, Naceur, Christophe, Marc, Marie.

Je remercie Sylvain Pion, pour son aide très précieuse tout au long de mon travail.

Je remercie Pierre Alliez, pour son aide, et son suivi durant mon DEA. Pour tout ce que j'ai pu apprendre de lui.

Je tiens à remercier également le personnel administratif de l'INRIA, notamment Vanessa Wallet, pour son aide au niveau administratif.

Je remercie le personnel du CROUS de Nice pour leurs aides et leurs soutiens durant tout mon séjour à Nice, et spécialement Mme. Rodriguez pour tous ces efforts et son soutien.

Je remercie M. le consul d'Algérie à Nice, ainsi que tout le personnel du consulat d'Algérie à Nice. Je remercie spécialement M. Toufik Akkdache, et M. Hamdani pour leur soutien.

A ma tendre mère et à mon cher père pour leurs sacrifices en témoignage de toute mon affection

A mes sœurs Fatna et Fatima Zohra

A mes frères Benamar, Mohammed, Abderrahim, Youcef, et Mostefa.

A mes neveux et nièces

A tous mes amis en Algérie et en France, et ailleurs.

A tous ceux qui me sont chers.

A mon pays.. l'Algérie.

Table des matières

Table des figures

Les pseudocodes

Liste des tableaux

Résumé

Résumé

La modélisation des objets géométriques est incontournable dans de nombreuses disciplines et applications. L'évolution des moyens d'acquisition et de stockage a produit une hausse énorme des volumes utilisés pour stocker ces objets. La réduction des tailles de ces volumes fait l'objet de plusieurs domaines de recherches ; comme la compression, qui vise à compresser le volume au maximum, et l'élaboration de structures théoriques compactes qui minimisent la taille nécessaire à la représentation. Le but de cette thèse est de concevoir, et d'évaluer des solutions pratiques et exploitables pour représenter de façon compacte les triangulations. Pour ce faire, deux issues sont explorées : modifier la représentation interne en mémoire des objets géométriques, et redéfinir les types abstraits des objets géométriques correspondants. Une première solution consiste à utiliser des indices sur une taille arbitraire de bits, au lieu des références absolues. Les gains dépendent de la taille de la triangulation, et aussi de la taille du mot mémoire de la machine. Le handicap majeur est le coût élevé de la méthode en termes de temps d'exécution. Une deuxième piste consiste à utiliser des catalogues stables. L'idée consiste à regrouper les triangles dans des micro-triangulations, et de représenter la triangulation comme un ensemble de ces micro-triangulations. Le nombre des références multiples vers les sommets, et des références réciproques entre voisins est alors nettement réduit. Les résultats sont prometteurs, sachant que le temps d'exécution n'est pas dramatiquement altéré par la modification des méthodes d'accès aux triangles. Une troisième solution consiste à décomposer la triangulation en plusieurs sous-triangulations permettant ainsi de coder les références dans une sous-triangulation sur un nombre réduit de bits par rapport aux références absolues. Les résultats de cette techniques sont encourageants, et peuvent être amplifiés par d'autres techniques comme le codage relatif des références, ou le partage de l'information géométrique des sommets sur les bords entre les différentes sous-triangulations. L'élaboration de structures compactes nécessite encore plus d'intérêts, et plusieurs pistes sont à explorer pour pouvoir arriver à des solutions plus économiques en termes d'espace mémoire.

Mots-clés

Triangulations, représentations compactes, structures de données géométriques, indices, catalogues, sous-triangulations.

Abstract

Modeling geometric objects is inescapable in various domains and applications. The development of acquisition and storage tools has produced a huge increase of the volumes used to store these objects. Many disciplines aim to reduce the size of these volumes ; such as compression, which tries to compress the volume, and the elaboration of theoretical compact data structures that minimize the space required for the representation. The goal of this thesis is to design, and evaluate practical and workable solutions for compact representations of triangulations. To do this, two issues has to be explored : modify the internal memory representation of the geometric objects, and redefine the abstract types associated to the geometric objects. The first solution proposed uses indices of an arbitrary length, instead of absolute references. The gain depends on the size of the triangulation, and the length of the memory word too. The main disadvantage of this solution is the high cost in terms of running time. The second solution uses stable catalogs. The structure gathers triangles into packages, and represents the triangulation as a decomposition of such packages. The number of multiple references to the vertices, and the number of reciprocal references between neighbors are clearly reduced. The results are interesting, and the running time is not dramatically affected by the added indirection levels. The third solution decomposes the triangulation into many sub-triangulations. The references in a sub-triangulation are coded using a reduced number of bits in comparison with the absolute references. The results of this technique are promising, and could be improved using other methods such as the relative coding of references, and sharing common vertex data between different sub-triangulations. The elaboration of compact data structures needs more attention, and several issues have to be explored in order to improve the solutions proposed.

Key-words

Triangulations, compact representations, geometric data structures, indices, catalogues, sub-triangulations.

Introduction Générale

Introduction Générale

Les structures géométriques telles que les triangulations ont une grande importance dans une vaste gamme d'applications. On peut citer entre autres les modèles de calcul par éléments finis, la conception bidimensionnelle et tridimensionnelle des maquettes et des prototypes, et les simulations dans de nombreuses disciplines comme la physique et la médecine.

Les études théoriques de la géométrie algorithmique lors des dernières décennies ont permis un progrès énorme en termes de conception d'algorithmes, et de rétablissement de bornes théoriques très intéressantes, voir optimales. Cependant, la pratique en matières de conception et d'implantation de structures de données pratiques et efficaces n'a pas suivi ce progrès en rigueur.

En effet, avec l'évolution des technologies d'acquisition, et la multiplication continue des capacités de stockage, la nécessité d'avoir des structures de données capables de gérer les données immenses existantes devient primordiale pour certaines applications. La finalité recherchée dans ce contexte est bien évidemment la conception de structures non gourmandes en termes d'espace mémoire.

Jusqu'à présent, la plupart des applications et des travaux se sont axés sur deux types de solutions :

La compression des données

La compression au sens propre du terme signifie la tentative d'éliminer au maximum la redondance existante dans la structure. Les travaux dans cet axe ont pu réaliser des taux de compression très intéressants, tout en atteignant les limites théoriques. La compression transforme la structure de données en un code constitué généralement d'une suite de bits suivant un schéma précis permettant au décodeur d'en extraire la structure initiale. La compression produit donc un bloc non utilisable en soi. Le résultat est donc inexploitable, du fait qu'on ne peut pas accéder aux champs et aux données. Par conséquent, cette solution est pratique pour le stockage des données, et pour la transmission non progressive sur réseaux.

Les algorithmes à mémoires auxiliaires

Le traitement en mémoire secondaire, présume la non possibilité de traiter les données entièrement en mémoire vive. Ici, le but est de trouver des techniques bien adaptées pour pouvoir explorer les données et déterminer comment les partitionner, afin de minimiser le transfert entre les mémoires de masse et la mémoire vive lors du traitement. Ces modèles sont utilisés pour manipuler les gros jeux de données. Cette solution ne réduit pas les volumes des données, mais cherche à bien gérer ces volumes.

Le contexte de la thèse

Dans le cadre de l'ACI[1] *Masse de Données* lancée par *l'ANR*[2], plusieurs équipes de recherche se sont regroupées dans le projet *GéoComp* sous le thème de la compression de données de nature géométrique. L'objet du projet est de développer de nouveaux outils de représentation implicite compacte adaptative qui soient efficaces vis-à-vis des requêtes de base pour la géométrie. Cette thèse est financée en partie par ce projet, et contribue à ses objectifs.

La bibliothèque CGAL

Une partie des expérimentations de notre travail concerne la bibliothèque *CGAL*, et tente de fournir des composants qui soient inclus dans le futur dans cette bibliothèque.

La bibliothèque *CGAL* (*Computational Geometry Algorithms Library*)[CGAL] est le fruit de la collaboration entre plusieurs laboratoires de recherche, visant à rendre accessible un grand nombre d'algorithmes et de structures de données pour la géométrie algorithmique. La bibliothèque contient entre autres des structures et des algorithmes pour les triangulations, les diagrammes de Voronoï, la génération de maillages, l'approximation, et bien d'autres composants.

Ces algorithmes et structures opèrent sur des objets géométriques comme les points, les vecteurs, et les segments ; et manipulent des prédicats géométriques, utilisant différents types d'arithmétiques pour le calcul. Le but est de fournir à l'utilisateur des algorithmes qui soient robustes et exacts.

Contributions

Le but de cette thèse est de suivre une autre piste qui n'est pas assez traitée dans la littérature. Les structures de données en pratique sont souvent conçues pour avoir une souplesse et une efficacité en termes d'utilisation. Toutefois, l'aspect espace (la quantité de

[1] Action Concertée Incitative.
[2] Agence Nationale de la Recherche.

mémoire allouée par le programme) n'est pas bien considéré. L'élaboration de structures de données pratiques, permettant la manipulation de gros jeux de données, et évitant ainsi le plus longtemps possible le transfert entre les mémoires externes et la mémoire vive est un enjeu qui mérite d'être entrepris.

Des travaux théoriques ont déjà été faits, et des résultats intéressants ont été présentés, mais n'ont pas encore été mis à l'épreuve de la pratique. Dans ce qui suit, ces solutions, et d'autres, seront étudiées, en termes de rendement mémoire, mais aussi sur le plan efficacité, et influence sur la complexité de la mise en œvre et le temps de calcul.

La première étape consiste à exploiter la limite entropique pour la représentation des références de n objets, en remplaçant les pointeurs vers les objets de la triangulation par des références codées au bit près sur la taille optimale $\log(n)$[3] bits, où n est le nombre de ces objets dans la triangulation. Et d'en étudier les implications pratiques sur la complexité et le temps de calcul.

La seconde étape consiste à chercher des petites structures redondantes dans la triangulation, et de représenter la triangulation comme une décomposition en ces objets qui sont les catalogues. Ceci est une extension, et un passage en pratique des résultats déjà étudiés en théorie, concernant la représentation hiérarchique des triangulations avec l'utilisation des catalogues. Le but de cette indirection est de minimiser le nombre de références multiples des triangles vers les sommets, et le nombre de références réciproques entre les triangles.

Une dernière étape consiste au passage à des références de petites tailles pour minimiser le coût global. Ce but est atteint en subdivisant la triangulation en petites sous-triangulations, permettant ainsi d'utiliser des références locales sur une taille réduite de bits. La structure est alors découpée en plusieurs structures de tailles moyennes, pour avoir des pointeurs locaux plus petits.

Les travaux du premier axe concernent les triangulations en dimension 3. Cependant, le reste de la thèse porte sur les triangulations en dimension 2.

L'utilisation des pointeurs sur une taille optimale de bits est exploitable en dimension 2 et 3, et le passage de la dimension 3 à la dimension 2 est direct. L'extension des autres structures proposées dans cette thèse vers les dimensions supérieures n'étant pas triviale est laissée à des travaux futurs, et n'est pas couverte par cette thèse. Cependant, la subdivision des structures globales pour avoir des structures redondantes, ou pour avoir des références plus petites en dimension 3 est une piste qui mérite d'être exploitée.

[3] voir les notes aux lecteurs en fin de l'introduction

Organisation de la thèse

La thèse est organisée en deux parties. La première partie comprend deux chapitres. Un chapitre de préliminaires, couvrant les définitions de base des notions utilisées dans le reste de la thèse, ainsi que les algorithmes, et les notions fondamentales de structures de données. Un deuxième chapitre est consacré à l'état de l'art. Ce chapitre couvre différents thèmes voisins, les structures de données pour la représentation des triangulations, la compression des maillages et de triangulations, et les structures de données compactes, ainsi qu'une brève description des algorithmes de traitement en mémoire auxiliaire.

La deuxième partie couvre le travail réalisé pendant la durée de la thèse, réparti en trois chapitres. Le premier chapitre discute l'utilisation des indices sur une taille optimale de bits au lieu des références absolues pour représenter les références des objets de la triangulation. Le deuxième chapitre présente une structure à base de catalogues stables pour représenter les triangulations. Le troisième chapitre présente une structure à deux niveaux permettant une réduction considérable de la taille des références pour coder les triangulations.

Notes aux lecteurs

- Dans cette thèse, le symbole log réfère toujours au logarithme de base deux, vu que c'est le seul logarithme utilisé.
- Le codage des références de n éléments nécessite des mots dont la taille est égale à :

$$\begin{cases} \log(n)\ bits & \text{si } \log(n) \text{ est un entier} \\ \lceil \log(n) \rceil + 1\ bits & \text{sinon} \end{cases}$$

  Pour simplifier la notation, cette taille est dénotée par $\log(n)\ bits$ dans la suite de cette thèse.

Première partie

Préliminaires

Chapitre 1

Définitions et préliminaires

1.1 Quelques notions de la géométrie algorithmique

Cette section présente certaines notions de base de la géométrie algorithmique et de la théorie des graphes sans trop rentrer dans les détails. Le lecteur peut s'adresser aux références pour une vue plus approfondie.

1.1.1 Les graphes

Un graphe $\mathcal{G}$ est défini par une paire d'ensembles $(\mathcal{A}, \mathcal{S})$, tels que les éléments de $\mathcal{A}$ sont des paires (s_0, s_1), où s_0 et s_1 sont des éléments de S. Les éléments de $\mathcal{S}$ sont appelés *sommets*, *noeuds*, ou *sites* ; et les éléments de $\mathcal{A}$ sont appelés *arêtes*[Bond 76, Dies 00].

Un graphe planaire est un graphe qui admet un *plongement* dans le plan : ce qui revient à dire que le graphe peut être dessiné dans le plan sans que ses arêtes ne s'intersectent sauf aux extrémités[Bond 76].

Un graphe connecté est un graphe $\mathcal{G}$, où pour toute paire de ses sommets (v, w), il existe au moins un chemin reliant les deux sites v et w[Dies 00].

La formule d'Euler-Poincaré s'applique a tout graphe connecté planaire $\mathcal{G}$, à s sommets, a arêtes, et f faces[Bond 76, Dies 00], et est la suivante :

$$s - a + f = 2 \tag{1.1}$$

Une carte est une extension de la notion de graphe, incluant la notion de face. Une carte résulte alors du plongement d'un graphe dans une surface.

1.1.2 Les triangulations

Une triangulation est une subdivision planaire où toutes les faces sont des triangles[Prep 85]. Une triangulation d'un ensemble fini $\mathcal{P}$ de points est un graphe planaire dont les sommets sont les points de $\mathcal{P}$, avec un nombre maximal d'arêtes[Berg 00]. Ce qui revient à relier tous les point de $\mathcal{P}$ entre eux avec des arêtes sans aucune intersection[Prep 85, Chen 96].

Simplexes et Complexes

La triangulation est donc définie par un ensemble de points, appelés aussi des *0-simplexes*, où *simplexes d'ordre 0*. Pour pouvoir exploiter efficacement la triangulation, les structures de données associées utilisent les simplexes d'ordre supérieurs, qui sont les simplexes d'ordre 1, ou les arêtes, les simplexes d'ordre 2, ou les triangles (appelés aussi *faces* en dimension 2 et facettes en dimension 3), et les simplexes d'ordre 3 pour les tétraèdrisations, ou les tétraèdres (appelés aussi *cellules*)[Bois 95].

Une triangulation en dimension 2 est alors un ensemble de simplexes de dimension 0, 1, et 2 ; et est alors appelée ***complexe simplicial d'ordre 2***. Une triangulation en dimension 3 (ou une tétraèdrisation) est aussi appelée ***complexe simplicial d'ordre 3***.

En général, un complexe $\mathcal{C}$ est un ensemble de simplexes vérifiant :

- tout sous-simplexe d'un simplexe de $\mathcal{C}$ est un simplexe de $\mathcal{C}$;
- deux simplexes de $\mathcal{C}$ ne s'intersectent pas, ou s'intersectent selon un simplexe de dimension inférieure.

Un complexe pur est un complexe $\mathcal{C}$ où tout simplexe de $\mathcal{C}$ est un sous-simplexe d'un simplexe de dimension maximale (voir figure 1.1).

Un simplexe singulier est un simplexe s d'un complexe $\mathcal{C}$ de dimension 2 (respectivement de dimension 3) ayant un voisinage (les simplexes incidents à ce simplexe) qui n'est pas homéomorphe à un disque (respectivement une sphère).

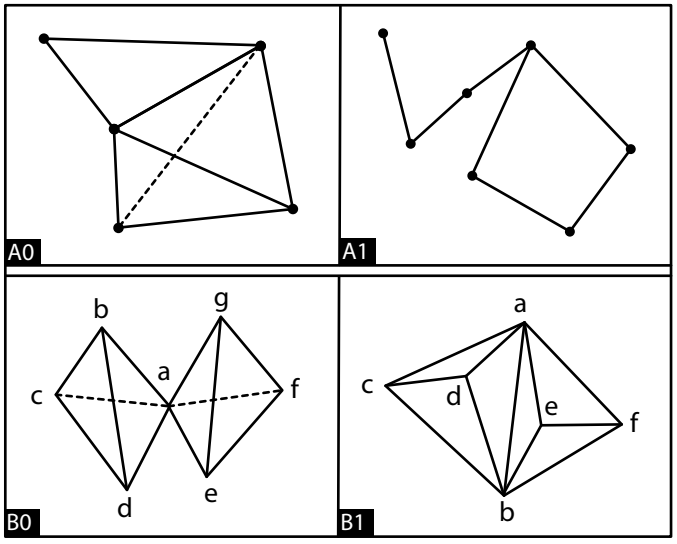

FIG. 1.1: Deux exemples de complexes non purs (A0) en dimension 3 (A1) en dimension 2 : Deux exemples de simplexes singuliers (B0) le sommet a est singulier (B1) l'arête (ab) est singulière[Bois 95].

Les maillages sont définis comme des plongements de graphes pour décrire des surfaces (*maillages surfaciques*) ou des volumes (*maillages volumiques*). Ceci revient à associer

des coordonnées géométriques aux sommets du graphe.

Le genre d'une surface est le nombre d'anses dans cette surface, et est égal à 0 pour les surfaces homéomorphes à une sphère.

Un maillage est **une variété** (ou *manifold*) si toute arête est incidente à exactement 2 faces, et tout sommet a un voisinage homéomorphe à un disque. Il est **avec bord**, si pour un nombre de cycles disjoints d'arêtes incidentes à exactement une face, les deux propriétés précédentes ne sont pas vérifiées. Il est **simple** s'il est de genre 0.

1.1.3 Arbre couvrant

Un graphe connecté, acyclique (sans cycles), et non orient est appelé ***arbre***. Un ensemble d'arbres est appelé ***forêt***. On appelle ***arbre de parcours*** ou ***arbre couvrant*** d'un graphe connecté $\mathcal{G}$, un sous-graphe de $\mathcal{G}$ qui est un arbre, et contenant tous les sommets de $\mathcal{G}$ [Deo 04, Corm 01].

Il existe plusieurs stratégies pour parcourir un arbre : visiter tous les nœuds de cet arbre.

Parcours en largeur d'abord Soit un arbre $\mathcal{G}$, et un sommet r désigné comme racine. Le parcours en largeur d'abord consiste à explorer l'arbre à partir de la racine, en visitant tous les sommets voisins à cette racine. Ces sommets visités sont ensuite considérés comme des racines, et le parcours continue (voir figure 1.2 A).

Parcours en profondeur d'abord A partir d'une racine r, le sous arbre défini par chacun des sommets voisins à r est visité en entier, avant de passer au voisin suivant. Le parcours descend dans l'arbre jusqu'à ce qu'il rencontre une feuille, et il remonte ensuite, en parcourant à chaque fois les branches non encore explorés (voir figure 1.2 B).

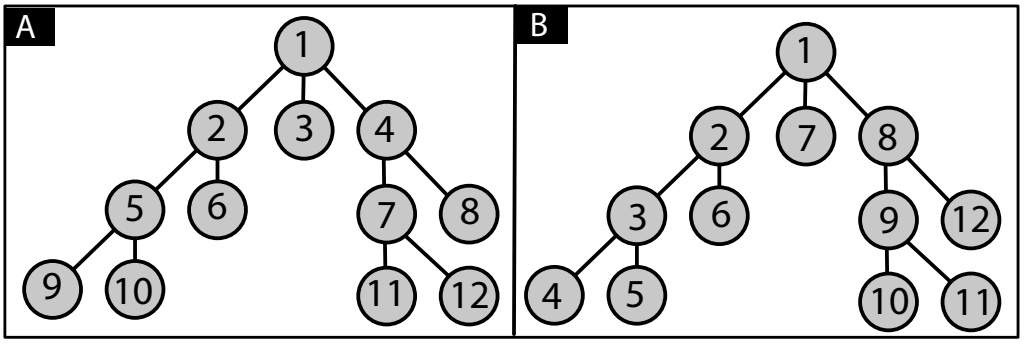

FIG. 1.2: Parcours d'un arbre en deux modes (A) en largeur d'abord (B) en profondeur d'abord

1.2 La triangulation de Delaunay

Une triangulation de Delaunay d'un ensemble de points $\mathcal{S}$, est une triangulation qui vérifie la propriété du cercle vide pour tous ses triangles : le cercle circonscrit à tout triangle ne contient aucun point de $\mathcal{S}$ en son intérieur strict. Cette triangulation est le graphe dual d'un diagramme de Voronoï, dont les centres sont les points de $\mathcal{S}$. La triangulation de Delaunay et le diagramme de Voronoï ont été largement étudiés en géométrie algorithmique [Bois 95, Berg 00, Chen 96, Prep 85].

1.2.1 Calcul de la triangulation de Delaunay

La construction de la triangulation de Delaunay peut être statique ou dynamique. L'algorithme présenté ci-dessous est dynamique, et est l'algorithme utilisé dans la suite de cette thèse. On trouve dans la littérature plus de détails sur les autres approches de calcul [Bois 95, Berg 00, Chen 96, Prep 85].

Le calcul se fait incrémentalement : à chaque insertion, la triangulation est modifiée pour maintenir la propriété du cercle vide valide partout.

1.2.2 Algorithme de construction par insertion et bascules d'arêtes

Pour insérer un nouveau point p dans une triangulation de Delaunay $\mathcal{T}$, cet algorithme procède comme suit :

Localiser le point p

Cette phase consiste à parcourir la triangulation à la recherche du triangle contenant le point à insérer. Si le point ne coïncide avec aucun sommet de la triangulation, le résultat de cette recherche est :

- soit un triangle t, si le point est dans ce triangle.
- soit une arête a, si la position du point est alignée avec les extrémités de l'arête.
- soit la face infinie, quand le point est en dehors de l'enveloppe convexe des points déjà insérés dans la triangulation.

La localisation est faite avec une marche par visibilité[Devi 01], vu que les triangulations calculées dans la suite sont toutes de Delaunay.

Insérer le point p

Insérer le point suivant le cas (voir figure1.3) :

- Dans le triangle t, en reliant le nouveau point p aux sommets du triangle par des nouvelles arêtes.

– Sur l'arête a, en reliant le nouveau point aux deux sommets opposés. Cette insertion est réalisée en pratique en deux étapes : insérer le point dans l'un des deux triangles incidents à l'arête, effectuer ensuite une bascule d'arête pour rétablir le bon cas de figure.
– Si le point se situe en dehors de l'enveloppe convexe, le relier aux points visibles de l'enveloppe convexe pour construire de nouveaux triangles.

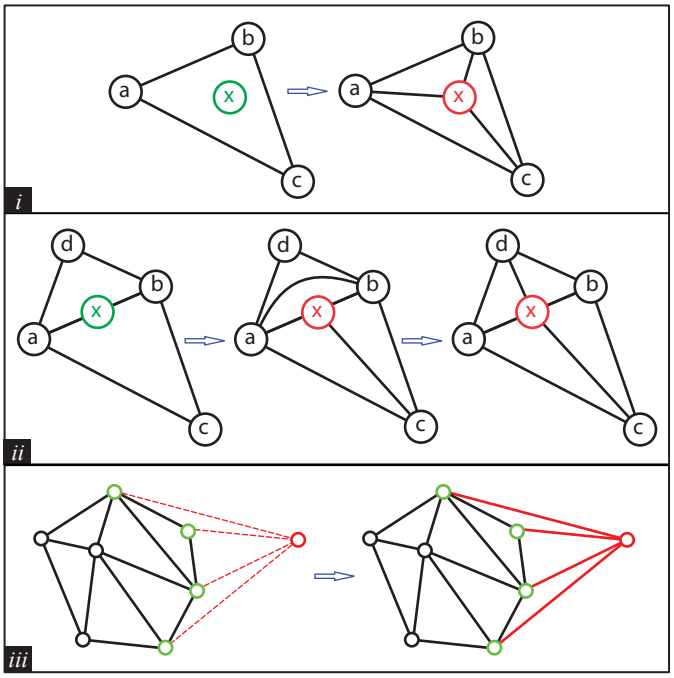

FIG. 1.3: (i) L'insertion combinatoire d'un point dans un triangle se fait en le reliant à tous les sommets de ce triangle. (ii) L'insertion d'un point sur une arête se fait en combinant deux opérations : (a) L'insertion de ce point dans l'un des triangles adjacents à l'arête (b) La bascule de l'arête en question pour obtenir la bonne configuration. (iii) Si le point (en rouge) est en dehors de l'enveloppe convexe, les sommets visibles sont retrouvés (en vert), et reliés au nouveau point par des nouvelles arêtes (en rouge).

Propager la propriété de Delaunay

Propager la propriété de Delaunay : en effectuant une série de bascules d'arêtes autour du nouveau sommet inséré, la propriété de Delaunay est alors établie sur toute la triangulation.

1.3 Les structures de données pour les triangulations

Une structure de données est une structure de stockage pour les données, munie des méthodes de création, de modification, et d'accès à ces données[Lasz 95]. C'est donc l'ensemble de ces trois modules :

- Un ensemble d'opérations pour gérer des types spécifiques d'objets abstraits. A ce niveau, ce sont les *types abstraits de données* qui sont définis.
- Une structure de stockage dans laquelle ces objets abstraits sont gardés en mémoire. Ce stade représente la configuration interne des données en termes d'occupation, d'allocation, et de restitution de mémoire.
- Une implantation de chaque opération en termes de structure de stockage.

Une triangulation est généralement représentée en mémoire par deux tableaux (souvent contigus), l'un pour les sommets, et l'autre pour les objets topologiques (les simplexes choisis pour modéliser la triangulation comme les triangles ou les arêtes).

Les tableaux utilisés sont souvent des tableaux contigus spécifiques aux différents langages de programmation, comme les conteneurs de la bibliothèque standard C++ STL[Stan 05], soit les vecteurs ou les listes, ou des conteneurs développés dans le même esprit[CGAL].

1.3.1 Séparation entre la topologie et la géométrie

La topologie d'une triangulation est décrite par le graphe associé aux sommets et aux arêtes de la triangulation[Hjel 06]. C'est donc l'ensemble des relations d'incidence entre les simplexes de la triangulation, qui sont les sommets, les arêtes, et les faces. Tandis que la géométrie de la triangulation représente le plongement de ce graphe en associant des coordonnées géométriques aux sommets de la triangulation.

La modélisation de la topologie de la triangulation repose souvent sur les méthodes développées en théorie des graphes, et c'est l'objet du chapitre suivant. La représentation de la géométrie est restreinte à la représentation des coordonnées des points, ce qui relève de la précision numérique, et de la représentation des nombres[Pion 99].

1.3.2 Calcul et mise à jour de la triangulation

Le calcul d'une triangulation d'un ensemble de points, et sa mise à jour peuvent être réalisés en utilisant trois opérations élémentaires :

- Insertion d'un nouveau point dans la triangulation.
- Suppression d'un sommet de degré trois de la triangulation.
- Bascule d'une arête de la triangulation.

Les structures de données proposées dans le reste de cette thèse supportent ces trois opérations élémentaires.

Chapitre 2

État de l'art

Les structures de données, et notamment les structures de données géométriques sont utilisées dans de nombreux domaines allant de la conception assistée par ordinateur et du calcul par éléments finis[Frey 00], à la modélisation des terrains, et à la modélisation tridimensionnelle, en passant par toute une gamme d'applications couvrant différentes disciplines telle que le graphisme[Zach 03], la modélisation, et la théorie des graphes.

Les travaux de base ont visé la conception de structures de données[Meht 04] qui soient efficaces, rapides et simples, tout en respectant les exigences des différentes types et objectifs des applications[Good 00]. Ce compromis n'est pas toujours facile à gérer, car les applications ne sont pas toujours catégorisées. L'efficacité en termes de temps d'exécution est demandée pour les applications en temps réel, où le besoin de réduire l'espace utilisé en mémoire passe généralement en second plan. Tandis que l'économie en matière d'occupation mémoire se pose pour le traitement des gros modèles, et induit dans la plupart des cas une diminution d'efficacité et une perte de simplicité en termes de mode d'accès aux éléments de la structure en ajoutant des niveaux d'indirection.

Nous allons voir dans ce chapitre une description des différentes solutions qui ont été proposées pour la réalisation de structures de données géométriques visant à répondre aux différentes exigences imposées par la variété des applications. Allant de simples structures de données, explicites et conçues pour des petits jeux de données, aux structures de données compactes ou succinctes. Nous allons voir aussi les grandes lignes des algorithmes de compression, et les structures de données proposées pour les applications faisant appel aux mémoires auxiliaires.

2.1 Les structures de données à base d'arêtes

Cette partie est consacrée aux modes de représentations qui ont été proposées pour les modèles polygonaux en général, et pas seulement les triangulations, et dont l'objet de base est l'arête, ou la demi-arête. Ces structures se sont développées autour de la modélisation

des solides et des surfaces[Mant 87]. Dans ce type de structures, l'information topologique est contenue principalement dans les arêtes de l'objet.

2.1.1 La structure de l'arête ailée

La première structure à base d'arêtes a été proposée par Baumgart[Baum 72, Baum 75], qui proposa une structure qu'il nomma *arête ailée* (voir figure 2.1).

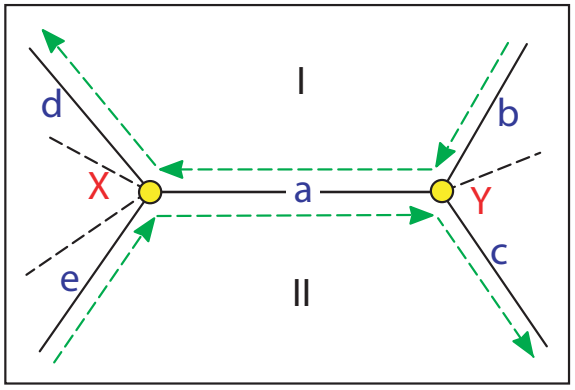

FIG. 2.1: La structure de l'arête ailée : L'objet de base est l'arête. Pour représenter l'arête a, ces informations sont utiles :(i) Les sommets de cette arête X et Y (ii) Les deux faces incidentes I et II (iii) Les prédécesseurs et les successeurs pour cette arête dans les deux faces : b, c, d, et e.

La structure garde un tableau de sommets avec leurs coordonnées pour la géométrie du modèle. Pour la topologie, elle dispose de trois types de données :

- **La face** qui ne garde qu'un pointeur vers une arête.
- **L'arête** qui garde huit pointeurs :
  - deux pointeurs vers les deux faces incidentes.
  - deux pointeurs vers les deux sommets incidents.
  - quatre pointeurs vers les arêtes prédécesseurs et successeurs dans les deux faces (voir la figure 2.1). Ces quatre pointeurs sont appelés ***les ailes*** de l'arêtes, d'où vient la dénomination ***arête ailée***.
- **Le sommet** qui ne garde qu'un pointeur vers une arête incidente.

Cette structure permet l'itération sur les sommets, les arêtes, et les faces ; ainsi que la circulation autour de ces trois complexes. L'auteur signale aussi l'efficacité de cette structure pour les applications nécessitant des fragmentations de faces ou d'arêtes[Baum 74].

Le nombre d'arêtes dans une triangulation d'une sphère topologique de n points et $2n$ triangles est égal à $3n$. Ce qui induit un coût global pour représenter une telle triangulation égal à $2n * 1 + 3n * 8 + n * 1 = 27n$ références. Ce qui est excessivement grand par rapport à une représentation à base de triangles explicite comme nous verrons plus loin.

2.1.2 La structure DCEL

Pour calculer l'intersection de deux polyèdres convexes, Preparata et Muller[Mull 78] ont proposé une structure à base d'arêtes vue comme une liste doublement connectée de nœuds, qu'ils appellent *Doubly-Connected-Edge-List* (Liste d'arêtes doublement connectée) ou *DCEL*.

La structure est conçue pour la représentation des subdivisions planaires[Prep 85], et l'objet de base dans la structure est l'arête, et chaque arête a est représentée par un nœud contenant six champs (voir figure 2.2) :

- Le sommet de départ de l'arête $v1$.
- Le sommet d'arrivée de l'arête $v2$.
- La face droite à l'arête $f1$.
- La face gauche à l'arête $f2$.
- Un pointeur vers l'arête suivante (dans le sens direct choisi) en tournant autour du sommet $v1$, en partant de l'arête courante.
- Un pointeur vers l'arête suivante (dans le sens direct choisi) en tournant autour du sommet $v2$, en partant de l'arête courante.

Le coût de représentation d'une telle structure est égal à $6 * 3n = 18n$ références pour une triangulation de n sommets.

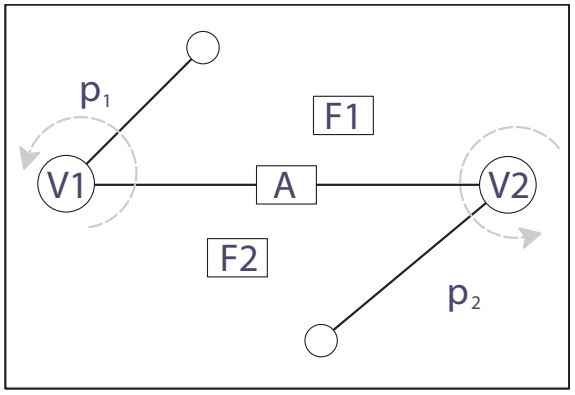

FIG. 2.2: La structure DCEL : L'objet de base est l'arête. Pour représenter l'arête a, ces informations sont utiles :(i) Les sommets de cette arête $v1$ et $v2$ (ii) Les deux faces incidentes $F1$ et $F2$ (iii) Les prédécesseurs et les successeurs pour cette arête autour des deux sommets dans le sens direct : $p1$, $p2$.

2.1.3 La structure Quad-Edge

Guibas et al.[Guib 83] proposent une structure complètement à base d'arêtes, pour modéliser à la fois une subdivision de l'espace, et sa duale (notamment la triangulation de Delaunay et le diagramme de Voronoï).

L'idée de base est de représenter chaque arête e de la structure par quatre *quad-edges e[i] (i=0..3)* ; les deux quad-edges $e[0]$ et $e[1]$ sont les deux versions orientées de e, et les deux quad-edges $e[2]$ et $e[3]$ sont les deux versions orientées de l'arête duale de e. Pour chaque quad-edge $e[i]$, un champ *suivant* est ajouté permettant de tourner autour du même sommet dans le sens direct choisi. Des primitives sont introduites pour permettre la navigation dans la structure, telles que la rotation d'une arête pour passer de l'arête à sa duale, la symétrique pour passer de l'arête à son opposée, et deux primitives pour accéder à l'arête suivante et précédente dans le sens direct choisi (voir figure2.3).

Si on considère une implémentation minimale des arêtes, incluant trois informations pour l'arête, soit le sommet de départ, le sommet d'arrivée, et une face incidente. Le coût global pour une triangulation de n sommets et $2n$ triangles est égal à $3n * 4 * 3 = 36n$ références.

Le grand avantage de cette technique est bien évidemment la possibilité de manipuler la subdivision primaire, et duale en utilisant la même structure.

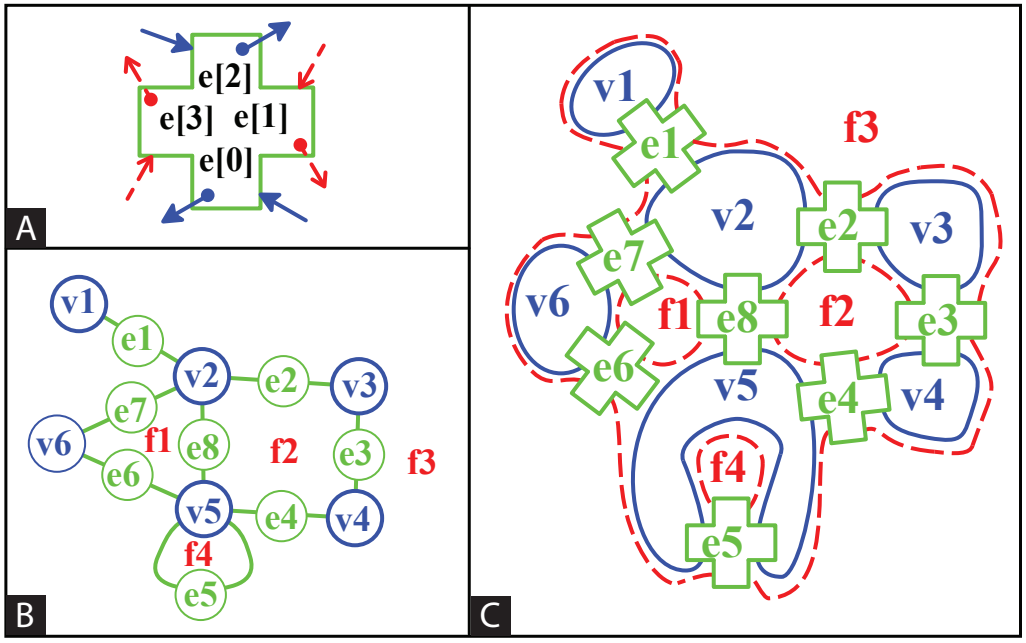

FIG. 2.3: La représentation quad-edge supporte la représentation de la subdivision et de sa duale (*i*) les quad-edges son en bleu (*ii*) les sommets sont en vert (*iii*) et les faces sont en rouge[Guib 83] : (A) un *quad-edge* avec les liens *suivant* et *précédent* (B) La subdivision à représenter (C) La structure Quad-edge associée.

2.1.4 La structure demi-arête

Kevin Weiler[Weil 85] propose une série de structures de données à base d'arêtes jugées adéquates pour la modélisation des objets en CAD (*Computer Aided Design*) et en CAM

(*Computer Aided Manufacturing*). Il présente notamment une étude sur la suffisance de ces structures à représenter des subdivisions (capacité à représenter toute la topologie, et à accéder à tous les simplexes de la structure).

Une variante de l'arête ailée

La première solution qu'il propose est une modification de la structure *arête ailée*, à laquelle il ajoute une information *side* à chaque référence d'arête pointée par l'arête courante (les références des quatre arêtes prédécesseurs et successeurs à l'arête courante). Cette information *side* indique de quel côté l'arête référencée est franchie (*side* doit toujours indiquer le côté de la face incidente à toutes les arêtes, voir figure 2.4). Ce surcoût en mémoire est justifié par la réduction de la complexité algorithmique pendant l'accès et le parcours des éléments.

Une structure Sommet-Arête

La seconde structure est *la structure* ***sommet-arête***. C'est une structure dont l'objet de base est la demi-arête. L'arête est alors subdivisée en deux demi-arêtes, chacune associée à une extrémité (qu'on appelle *sommet de référence*). Cette demi-arête se trouve alors adjacente à d'autres demi-arêtes autour d'un sommet. La représentation associe à chaque demi-arête les références suivantes :

- La demi-arête suivante en tournant autour du sommet de référence.
- Le sommet opposé au sommet de référence.
- Une face incidente.
- La demi-arête associée à l'autre sommet.
- Une autre information optionnelle qui est la demi-arête précédente.

Les notions de *suivante* et *précédente* sont liées au choix d'orientation autour du sommet de référence. Le coût global est égal à $3 * 5 = 15n$ références pour une triangulation de la sphère ayant n sommets.

Une structure Face-Arête

La troisième structure est *la structure* ***face-arête***. C'est aussi une structure dont l'objet de base est la demi-arête. L'arête est subdivisée en deux demi-arêtes, chacune associée à l'une des deux faces incidentes (qu'on appelle *faces de références*). Les informations associées à chaque demi-arête sont les suivantes :

- La demi-arête suivante dans la face de référence.
- Une référence vers un sommet incident.
- La face opposée à la face de référence.
- La demi-arête associée à l'autre face.

– Une information optionnelle qui est la demi-arête précédente.

Les notions de *suivante* et *précédente* sont liées au choix de l'orientation dans la face de référence.

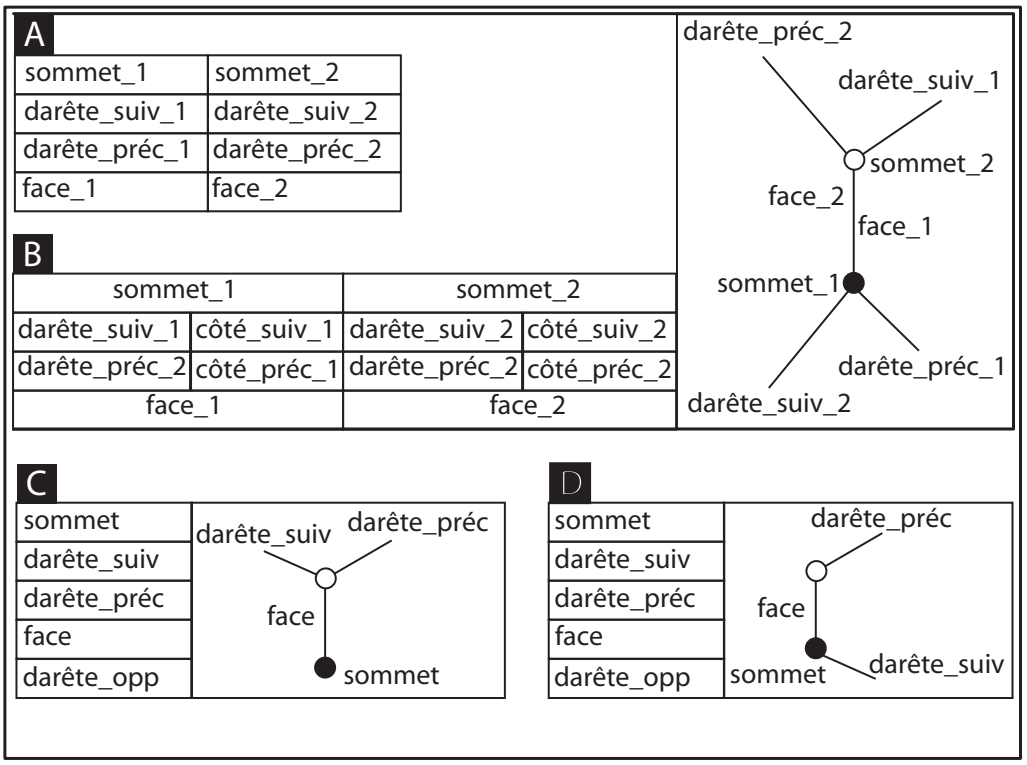

FIG. 2.4: La modélisation de l'arête dans les structures de données à base d'arêtes proposées par Kevin Weiler[Weil 85] : (A) La structure de l'*arête ailée* de Baugmart[Baum 75] (B) La version modifiée de l'*arête ailée* en ajoutant un entier à la référence de l'arête indiquant de quel côté de l'arête on est arrivé (C) La structure à base de demi-arêtes *Sommet-Arête* où la demi-arête est identifiée par son sommet de référence (D) La structure à base de demi-arêtes *Face-Arête* où la demi-arête est identifiée par sa face de référence.

Dans le cas des triangulations, les deux structures *face-arête* et *sommet-arête* se ressemblent, et induisent le même coût en mémoire. Néanmoins, le coût d'accès aux sommets et aux faces diffèrent. La structure *face-arête* modélise la face par un simple cycle de trois demi-arêtes, alors que la structure *sommet-arête* la modélise par un cycle de six demi-arêtes. Le parcours des arêtes incidentes à un sommet est par contre deux fois plus rapide dans la représentation *sommet-arête* que dans la représentation *face-arête*

2.1.5 Les n-GMaps

Lienhart[Lien 89] introduit un nouveau concept dans la représentation de la topologie de subdivisions de surfaces[Grif 76] en dimension n. Il s'agit des *n-G-maps* (où le n réfère à la dimension de la surface). Le concept de base dans cette représentation, largement

utilisé dans différents domaines, et notamment la modélisation géologique[Halb 99], est le *brin*. Ces brins correspondent aux occurrences des arêtes dans les facettes de la subdivisions. La structure de données est alors définie par l'ensemble des ces brins, et un ensemble d'opérateurs $\alpha_i, i = 1..n$ qui permettent de répondre aux requêtes, et de modifier la structure (voir figure 2.5).

Pour les triangulations, les brins sont définis comme des triplets (v_i, e_j, t_k), où v_i est un sommet de la triangulation, e_j est une arête de la triangulation, et t_k est un triangle de la triangulation. Les opérateurs α_i appliqués à un triplet d sont définis comme suit[Hjel 06] :

- $\alpha_0(d)$ permet l'accès à un triplet ayant la même arête et le même triangle, mais avec un sommet différent.
- $\alpha_1(d)$ permet l'accès à un triplet ayant le même sommet et le même triangle, mais avec une arête différente.
- $\alpha_2(d)$ permet l'accès à un triplet ayant le même sommet et la même arête, mais avec un triangle différent.

L'itération des opérateurs α_i permet le parcours de la triangulation et l'accès à tous les objets définis. Sur le bord de la triangulation, les arêtes sont considérées comme des points fixes de l'itérateur associé à l'opérateur α_2, vu que cet opérateur retourne la même arête.

Les triangles dans ce type de représentation sont définis implicitement comme étant des cycles de simplexes de dimensions inférieurs (soit les sommets et les arêtes).

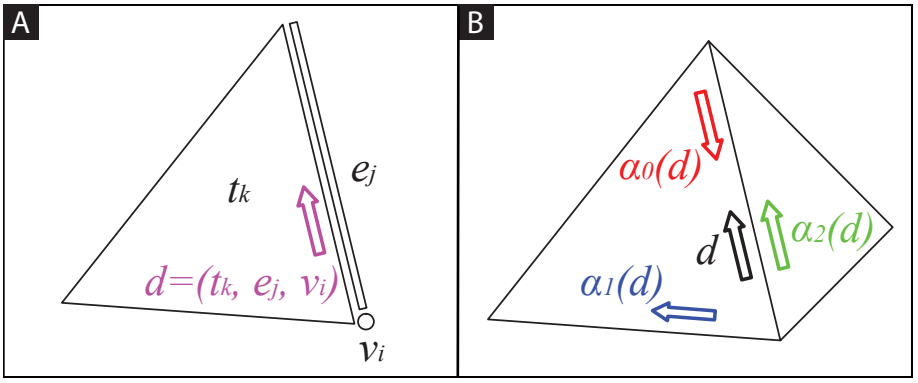

FIG. 2.5: La structure G-maps en dimension 2, les brins représentent l'objet de base dans cette représentation : *(a)* le brin d est un triplet (v_i, e_j, t_k) *(b)* les applications des fonctions α_0, α_1, et α_2 permettent le parcours de la structure.

2.1.6 La structure de l'*Arête orientée*

Campagna et al.[Camp 98] proposent une représentation similaire à celle utilisant les demi-arêtes de Weiler[Weil 85] dans laquelle l'objet de base est appelé *arête orientée*. Pour chaque arête orientée, on garde les informations suivantes :

- Le sommet de départ de l'arête.
- Le sommet d'arrivée.
- L'arête orientée précédente.
- L'arête orientée suivante.
- L'arête orientée voisine.

En plus, une référence à une arête orientée peut être ajoutée dans chaque sommet. La structure de données garde toutes les arêtes orientées de la triangulation dans un tableau, de sorte que chaque triangle soit représenté par les trois arêtes orientées dont les indices au tableau sont du type i, $i+1$, et $i=2$. Ce qui permet de représenter un triangle par un seul indice qui est l'indice de la première arête orientée dans le tableau.

Le nombre d'arêtes orientées dans une triangulation de n points étant égal à $6n$, cela conduit à un coût global de $2n + 6n * 5 + n = 33n$ références.

L'auteur propose une approche à plusieurs niveaux :

- La structure à taille complète : Celle décrite jusqu'ici.
- La structure à taille moyenne : Trois références seulement pour chaque arête orientée : Le sommet d'arrivée, La demi-arête voisine, et la demi-arête précédente. Cette structure induit un coût global de $19n$ références pour une triangulation de n points.
- La structure à taille réduite : Deux références sont stockées dans chaque arête orientée : Le sommet d'arrivée, et la demi-arête voisine. Le coût global est de $13n$ références pour une triangulation de n points. Avec la référence stockée dans chaque sommet, qui est la référence d'une demi-arête partante, le parcours de la topologie de la triangulation est garanti.

2.2 Les structures à base de triangles

Dans ce type de représentation, le concept de base est le triangle. La structure de base utilise deux tableaux : un pour les sommets où les coordonnées géométriques sont stockées ; le deuxième est pour les triangles, où on stocke les références des trois sommets qui définissent ce triangle. Cette structure minimale requiert $6n$ références pour une triangulation de n points. Cependant, l'accès aux voisins ne peut se faire en temps constant.

Pour avoir plus de performance, on permet un accès en temps constant aux voisins. Les références des trois triangles voisins peuvent être ajouté à la structure du triangle, ce qui augmente le coût de stockage à $12n$ références. Et si une référence est ajoutée à chaque sommet pour pouvoir accéder à un triangle incident, le coût augmente encore à $13n$ références.

Ce type des représentation est largement utilisé en pratique[Shew 96, TRIANGLE, CGAL], vu qu'il permet un stockage minimal par comparaison aux autres stratégies, tout en gardant une réponse en temps constant aux requêtes d'incidence et de voisinage.

La capacité de représentation

En matière de pouvoir de représentation de subdivisions de plans, il existe une correspondance entre les subdivisions et les représentations. Les cartes planaires sont mieux décrites par les structures à base de demi-arêtes et par les $G-Maps$, vu que les degrés des faces ne sont pas bornés. Alors que les triangulations sont mieux décrites par les structures à base de triangles, puisqu'il est l'objet de base dans de nombreuses applications.

2.3 Les structures à base de sommets

Dans ce type de structures[Clin 84], la triangulation est représentée comme un graphe d'incidence entre les nœuds (les sommets) de la triangulation.

La structure est une liste de sommets, où chaque sommet garde son *degré* (le nombre de ses sommets voisins), la liste de ces voisins, et une marque indiquant si ce sommet est un sommet du bord ou non.

Vu que le degré moyen d'un sommet dans une triangulation de n points est égal à 6, le coût d'une telle structure est $7n$ références.

2.3.1 Star-Vertices

Star-vertices[Kall 01b] est une structures de données dont l'objet de base est le sommet, et qui est conçue pour représenter des maillages planaires arbitraires.

Cette structure est décrite dans la figure 2.6. L'objet de base est le *sommet*, et à chaque sommet v sont associés ces attributs :

- Les coordonnées du point en question.
- Les références de tous les sommets voisins à v.
- Plus un indice pour chaque voisin v' indiquant lequel des voisins de v', en l'occurrence v", forme une face avec v et v'.

La dernière information permet de parcourir les sommets et les arêtes d'une face de manière directe. Mais dans la structure de base, elle peut être omise sans perturber l'accès à l'information.

Les auteurs présentent en plus une généralisation du concept d'itérateur, qu'ils appellent *travel*, qui permet d'itérer sur les élément du maillage (en l'occurrence les sommets) pour faciliter à l'utilisateur le parcours du maillage, et de considérer une représentation implicite des faces.

Sachant que la moyenne des degrés des sommets dans une triangulation est estimée à 6, le coût global de cette structure pour la triangulation d'une sphère à n point est de $7n$ références.

Cette technique souffre d'un inconvénient majeur, qui est l'absence de l'objet de base :

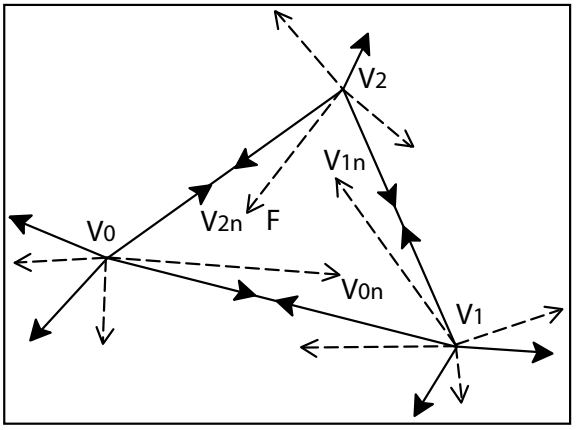

FIG. 2.6: Le diagramme de connectivité de la structure Star-Vertices[Kall 01b]. Les flèches représentent les références d'un sommet vers ses voisins, les flèches en pointillés, représentent les indices utilisés pour retrouver le sommet suivant dans la face incidente.

le triangle. Ce point ne se posait pas pour les auteurs de la méthode, car leurs subdivisions n'étaient pas que triangulaires[Kall 01a]. Un autre désavantage est le temps d'accès proportionnel au degré du sommet impliqué dans une opération. Ce coût peut être cher lorsque les degrés des sommets ne sont pas contrôlés.

2.4 Les triangulations en pratique

Cette section est consacrée aux solutions pratiques qui ont été mise en œuvre pour représenter les triangulations.

2.4.1 Une structure de données à base de demi-arêtes

En s'appuyant sur la programmation générique[Alex 01], adoptée par la bibliothèque C++ STL[Stan 05, Muss 95, Brey 97] ; Kettner [Kett 99] propose une conception logicielle d'une structure de données à base de demi-arêtes pour la modélisation des surfaces polyédriques sans singularités. La structure adoptée est comparable à la structure *face-arête* (voir section 2.1.4). La structure offre les méthodes d'accès suivantes :

- *suivante* pour accéder à la demi-arête suivante.
- *opposée* pour accéder à la demi-arête opposée.
- *précédente* pour accéder à la demi-arête précédente.

En plus, la structure garde pour chaque demi-arête un sommet incident qui est le sommet d'arrivée, et une face incidente qui est celle à gauche de la demi-arête, pour une orientation triangulaire directe.

La version qui a été introduite dans la bibliothèque *CGAL*[CGAL], repose sur deux concepts : La programmation en orienté objet, et la programmation générique en utilisant

les *templates* de C++[Vand 02].

La conception du composant de la bibliothèque CGAL[Kett 07] est illustrée dans la figure 2.7. Cette conception est un peu différente de celle présentée dans l'article original[Kett 99].

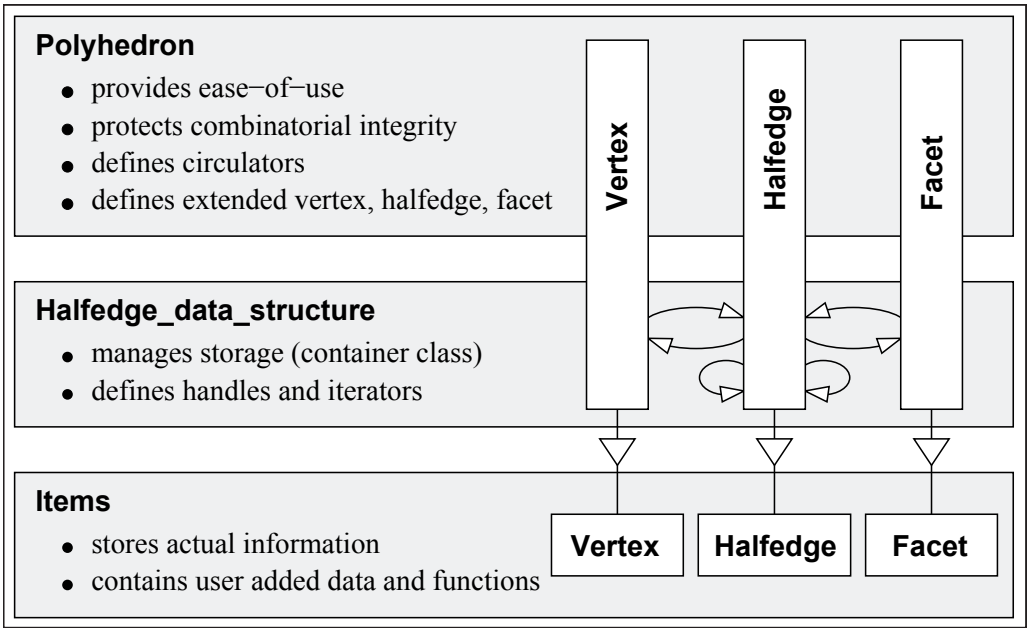

FIG. 2.7: La structure demi-arête de CGAL[Kett 99] : La programmation orientée objet, et la programmation générique permettent la séparation et la hiérarchie suivant les différents niveaux de conception.

Une caractéristique commune avec tous les composants de *CGAL*[CGAL 07], est la séparation entre la géométrie (les coordonnées des points et les informations associées), et la topologie.

La topologie est représentée par trois niveaux :

- Le niveau *objet* ; où sont définis les concepts correspondant aux objets réellement gardés en mémoire, avec toutes les définitions des méthodes d'accès et de mise à jour.
- Le niveau *structure de données* ; où la structure globale de stockage est définie, avec les méthodes d'insertion, de modification, et de suppression d'éléments. C'est cette partie qui se charge de l'allocation et de la restitution de la mémoire.
- Le niveau *Polyèdre* ; qui est le niveau *utilisateur*. C'est dans ce niveau là que sont définies les méthodes accessibles par l'utilisateur afin de conserver l'intégrité de la structure. Les *circulateurs* qui permettent de visiter tous les voisins d'un sommet, ou autour d'une face sont définis à ce niveau là.

2.4.2 Les triangulations dans *CGAL*

Dans le même contexte, l'implantation des triangulations dans la bibliothèque CGAL[Bois 02] est à base de triangles (voir section 2.2), et repose sur la séparation entre la géométrie et la topologie, et la programmation générique[CGAL 07, Fabr 00].

Plus de détails sur les triangulations de *CGAL* seront présentés dans le chapitre 3.

2.4.3 TTL (Template Triangulations Library)

Øyvind Hjelle propose une conception logicielle générique pour les triangulations[Hjel 00] basée sur la représentation GMaps[Lien 89]. Ce travail est l'extension d'une autre conception dédiée à la modélisation géologique[Halb 99].

La conception utilise les *templates*[Vand 02] de C++ pour définir une interface générique implantant les algorithmes et les méthodes de parcours et de modification, ainsi que les opérateurs α_i. Cette implantation prend comme paramètre *template* le brin (voir section 2.1.5), dont la définition, ainsi que les itérateurs sur les éléments de la triangulation (les sommets, les arêtes, et les triangles) sont laissés à l'utilisateur pour satisfaire les besoins et les spécificités de l'application.

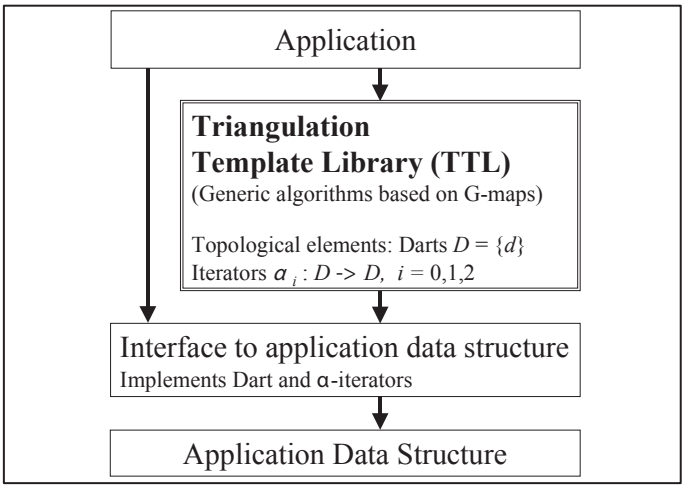

FIG. 2.8: La conception de la *Template Triangulations Library* proposée par Øyvind Hjelle[Hjel 00].

2.5 La compression des structures triangulaires

La compression des maillages (ou de triangulations) consiste à coder la triangulation en éliminant au maximum la redondance en maximisant l'entropie de la structure. La structure de données après la compression est inexploitable, et pour pouvoir manipuler la structure, ou accéder aux éléments, il faut décompresser toute la structure et reconstruire

la structure explicite.

Pour une étude détaillée et complète des techniques de compression de maillages et de triangulations, une vaste gamme de publications existe[Alli 05, Gand 01, Gumh 00, Gots 01, Isen, Lewi, Peng 05, Ross 03a, Ross 04].

Nous allons présenter dans cette section les différentes stratégies utilisées pour compresser les triangulations, en décrivant brièvement les algorithmes les plus connus dans chacune d'elle.

2.5.1 Séparation entre la topologie et la géométrie

Comme pour les structures de données explicites, la compression des structures géométriques sépare les deux aspects de la triangulation : la topologie, et la géométrie.
La plupart des algorithmes de compression sont définis suivant un schéma comprimant la topologie d'abord, vu que la topologie occupe la plus grande partie de la structure. Cependant, après les progrès réalisés sur cette partie, la partie dominante après codage est souvent la géométrie. C'est pourquoi une attention plus grande a été portée sur la compression de la géométrie des maillages dans des travaux récents[Peng 05, Alex 01].

2.5.2 Codage de la topologie

Codage des graphes planaires

La compression des maillages surfaciques[Cast 06a, Isen 05c] (notamment des triangulations planaires) trouve ses racines dans le codage des graphes[Tutt 62, Tura 84, Keel 95]. Le principe est quasiment le même dans toutes les approches, et consiste à définir un ordre d'énumération sur les sommets (ou/et) les faces de la subdivision (triangulation ou autre) suivant un schéma de parcours ; le maillage est ainsi parcouru et un code unique est généré permettant au décodeur de reconstruire intégralement la topologie initiale.

Théoriquement, la borne inférieure pour la représentation d'une triangulation planaire est $3,24$ *bps* (bits par sommet)[Tutt 62]. Turan a fourni le premier algorithme permettant un codage en nombre constant de bits pour les graphes planaires[Tura 84] avec *4 bits par arête*, ou 12 *bits par sommet* pour les triangulations, qui peut être optimisé à 6 *bits par sommet*[Isen 05c]. Keeler et Westbrook[Keel 95] proposent un algorithme pour réaliser un codage de *3,58 bits par arête* pour les graphes planaires, et $4,6$ *bits par sommet* pour les triangulations.

Compression de maillages

Le passage du codage des graphes planaires à la compression des maillages a été inauguré par Taubin et Rossignac[Taub 98]. Le format comprimé du maillage est défini par deux arbres couvrants, l'un pour les sommets, l'autre pour les faces, codé en longueurs

de chemins. Les taux pratiques présentés sont autour de 4 *bits par sommet*. Les travaux suivant dans la compression des maillages peuvent être classés comme ceci :

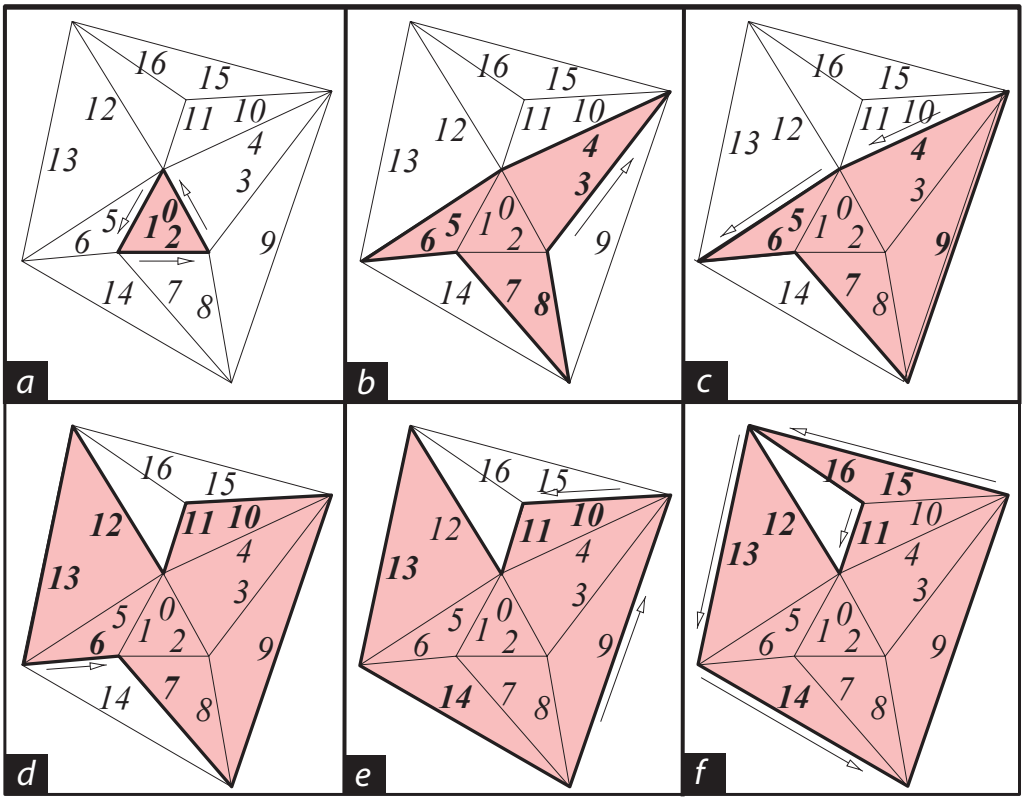

FIG. 2.9: Le parcours des faces d'un maillage en profondeur d'abord pour le codage de la traingulation[Gumh 98].

Les approches à base de triangles

Gumhold et Straßer[Gumh 98] proposent un parcours en largeur d'abord de la triangulation à partir d'un triangle choisi (voir figure 2.9). La triangulation est divisée en deux régions : *La région visitée*, et *la région à visiter*. Le parcours se poursuit en émettant un code associé à chaque cas de figure rencontré lors de la conquête d'un nouveau triangle. Un code supplémentaire est ajouté lorsqu'un *split* apparaît (lorsque le bord de la région conquise s'auto-intersecte, voir figure 2.10). Les résultats présentés vont de 4 *bits par sommet* pour les maillages simples à 8 *bits* lorsque le genre est élevé, ce qui est justifié par l'augmentation du nombre de *splits*.

Un des algorithmes de compression les plus connus a été proposé par Rossignac[Ross 99a] : *Edgebreaker*. L'algorithme consiste à visiter la triangulation en parcourant les triangles en profondeur d'abord (parcours de la triangulation en spirale), et à engendrer un code de sortie à cinq étiquettes *c-l-r-e-s* (dont une étiquette pour les *splits*) associant une étiquette à chaque triangle de manière à permettre au décodeur de reconstruire la to-

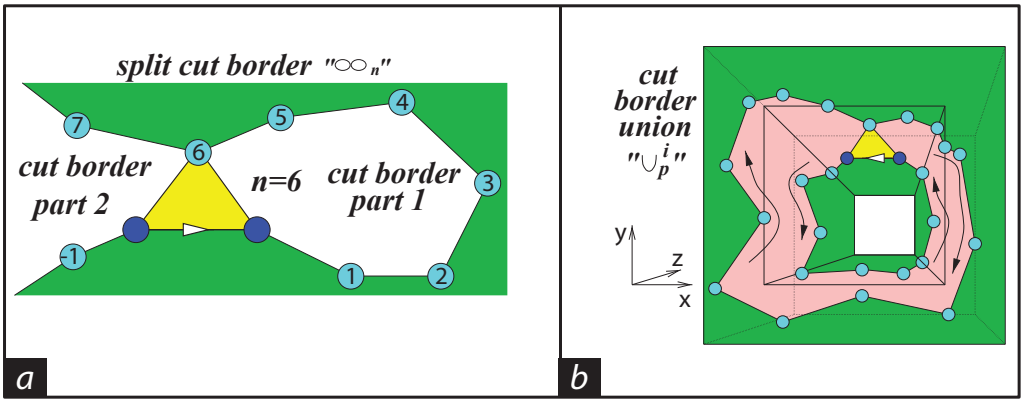

FIG. 2.10: Les *splits* dans le codage des triangulations : auto-intersection du bord de la région conquise lors du codage d'une triangulation[Gumh 98].

pologie exacte de la triangulation. Une nouvelle structure de données appelée *Corner Table*[Ross 01, Ross 03b] est utilisée pour représenter et modifier la triangulation au cours du traitement. Cette structure est la suivante :

- La géométrie est gardée dans un tableau où chaque entrée correspond à un sommet et stocke les trois coordonnées du sommet.
- Un triangle est défini par des références vers les trois sommets incidents. Le nombre de triangles est t. Les triangles sont stockés dans la même table V utilisée pour les sommets sous forme de trois *corners*.
- La topologie est gardée comme une liste de correspondances *Sommet-Triangle* dans une table V à $3t$ entrées : chaque entrée dans cette table représente un *corner* défini par un sommet de la triangulation et un triangle.
- Pour pouvoir parcourir la triangulation, les opérations *suivant* et *précédent* sont définies pour accéder aux corners suivant et précédent dans le même triangle à partir du troisième corner dans ce triangle.
- Des informations additionnelles peuvent être ajoutées ou *cachées* pour permettre un accès directe et efficace aux voisins : Le corner *opposé* à un corner donné, le corner *gauche* et *droit* d'un corner donné (voir figue 2.11).

Pour marquer les sommets et les triangles visités lors du parcours de la triangulation pendant la compression, un bit est ajouté pour chaque sommet et pour chaque triangle. Le parcours est effectué en profondeur d'abord en boucle, et marque les triangles et les sommets visités au fur et à mesure. Le triangle courant est repéré par son corner c opposé à l'arête qui a permis de l'affranchir lors du parcours. Suivant le cas qui se présente (en testant les marques des sommets -visités ou pas- et les marques des voisins), une étiquette est insérée dans le code de sortie (voir figure 2.12).

Plusieurs variantes et adaptations ont été proposées pour cette approche ; un pré-traitement du maillage pour le rendre adapté a la méthode[Atte 03], une adaptation pour les sur-

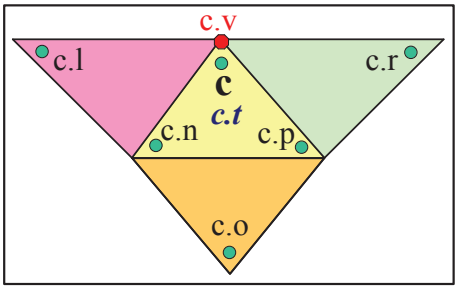

FIG. 2.11: La structure *Corner Table* représente chaque corner c par son sommet $c.v$, son corner suivant $c.n$, son corner précédent $c.p$, son corner gauche $c.l$, son corner droit $c.r$, et son corner opposé $c.o$[Ross 03b].

faces de genres différents de zéro[Lope 02], une généralisation pour les surfaces arbitraires (en genre et en nombre de composantes)[Lewi 04], une spécialisation pour les maillages réguliers[Szym 01], des optimisations apportées au décodeur[Ross 99b][Isen 05c], une généralisation pour les maillages tétraèdriques[Szym 00], et non-triangulaires[Kron 01]. Une étude des bornes et des taux réalisés de l'algorithme a été également publiée[King 99].

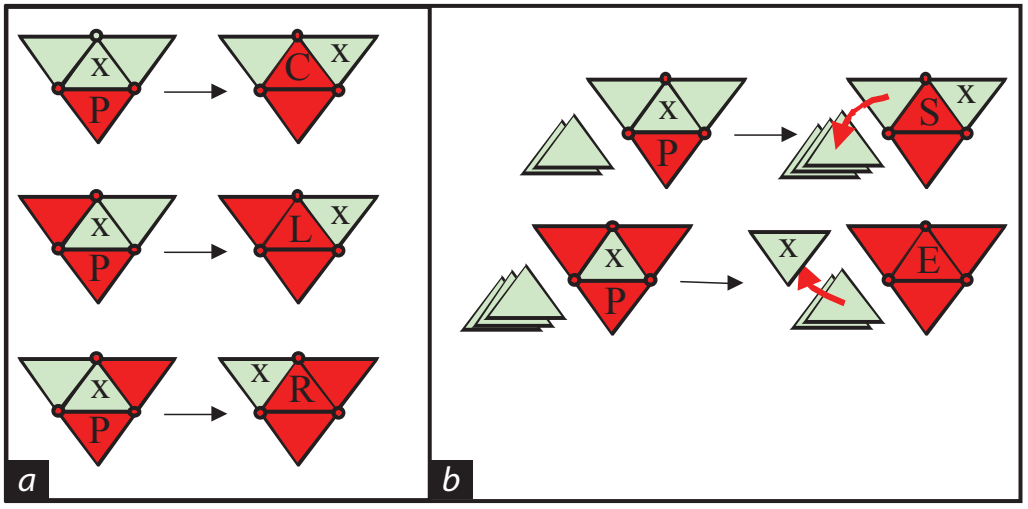

FIG. 2.12: Les règles de parcours du codeur *edgebreaker* exprimées comme des états : (a) Le triangle à visiter à une étape du parcours est représenté par x ; son *parent* dans l'arbre couvrant est P ; les sommets et les faces conquis sont colorés en rouge ; et le code émis est exprimé dans le triangle courant après son déplacement vers la zone conquise (b) Lorsque une auto-intersection apparaît dans le bord de la zone conquise, un triangle est mis dans la pile, et l'autre détermine le chemin à explorer, quand le chemin est traité en entier, le premier triangle est tiré de la pile, et le reste des triangles est traité[Ross 99b].

Les approches à base d'arêtes

Une approche basée sur le parcours du graphe dual de la triangulation[Li 98] consiste à parcourir les faces de la triangulation en profondeur d'abord. Les coûts sont en pratique estimés à 3 *bits par sommet* en moyenne. Un autre algorithme à base d'arêtes est proposé par Isenburg et al.[Isen 00a, Isen 00b], où une frontière est maintenue entre la région visitée et la région à visiter, et une arête dite *porte* réalise le passage des triangles entre les deux régions. L'algorithme utilise huit symboles pour coder les cas de figures lors de la mise à jour de la frontière. Les taux de compression pratiques présentés varient suivant les types de maillages : on obtient des coûts allant de 1 à 4 *bits par sommet*.

Les approches à base de sommets

Dans une telle approche, appelées aussi *Codage avec degrés des sommets*, un sommet du maillage est considéré comme pivot, autour duquel toutes les opérations sont entreprises. Dans[Toum 98], une liste active de sommets (un cycle de sommets du maillage) est maintenue, et sépare le maillage en deux régions : interne (conquise), et externe (à conquérir). Un sommet est choisi comme pivot dans cette liste, une fois ce sommet traité (après avoir conquis tous les sommets voisins et avoir généré le code associé au degré du sommet), un autre sommet est désigné comme pivot, et le précédent est transmis à la région interne. Un *offset* est le nombre de sommets séparant deux sommets dans la liste active dans le sens d'orientation direct choisi. Cet *offset* sert à repérer les positions des *splits* (les auto-intersections de la liste active) le cas échéant. Les coûts pratiques varient entre 1 et 2 *bits par sommet* pour les modèles présentés[Toum 98]. Cette méthode a été généralisée aux maillages polygonaux[Isen 02a, Isen 02c], et aux maillages volumiques hexaèdriques[Isen 02b, Isen 03a]. Une étude théorique sur une version modifiée de l'algorithme est aussi disponible[Alli 01b].

Codage géométrique progressif

Gandoin[Gand 01] a proposé un ensemble de méthodes de compression qui donnent la priorité à la géométrie, c'est à dire les coordonnées des sommets. Ces méthodes sont progressives et sans perte d'information et adaptées à la transmission de données (voir figure 2.13). Ces méthodes sont adaptées à une large classe de structures géométriques non nécessairement triangulaires et généralisables à n'importe quelle dimension.

Le principe de l'algorithme de base proposé est de définir des cellules contenant les points à coder. En dimension 1, 2, et 3, les cellules sont respectivement le segment de droite, le carré, et le cube. En dimension 2, la cellule doit être subdivisée deux fois, et récursivement. Un ordre de subdivision doit être choisi et fixé afin que le codeur et le décodeur puissent communiquer. La figure 2.13 montre un exemple de l'algorithme. Cet algorithme permet d'obtenir un taux moyen de compression de $3,6$ *bits par sommets* pour

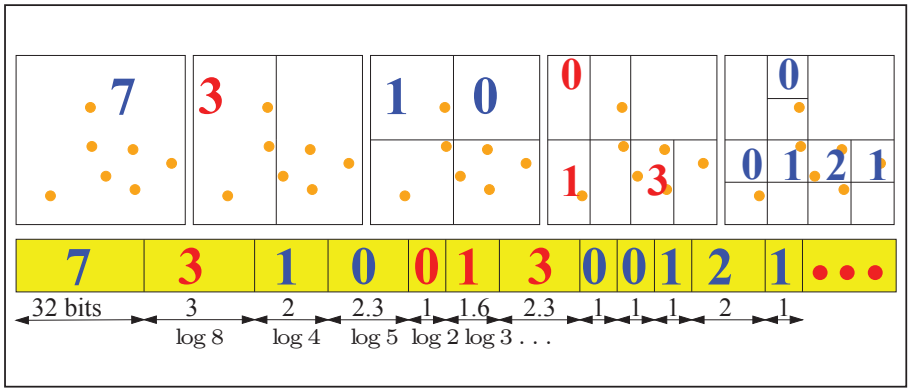

FIG. 2.13: Compression progressive sans perte : La première étape consiste à coder les coordonnées des points sur un nombre arbitraire de bits (32 par exemple). La deuxième étape subdivise récursivement les cellules en deux, et code le nombre de points contenus dans l'une des deux cellules sur un nombre de bits optimal. L'algorithme termine lorsqu'il n'y a plus de cellules subdivisibles. La sortie de l'algorithme est la suite des nombres de points situés sur les cellules successives.[Gand 02].

les maillages triangulaires surfaciques.

Autres approches

Le codage progressif[Hopp 96] ou multi-résolution[EDan 05] vise à compresser le maillage tout en produisant des niveaux de détails permettant de passer d'une représentation éparse à des représentations plus détaillés. Ces techniques sont souvent combinées à des techniques de simplification[Alli 01a, Cohe 99, Khod 00, Paja 00].

Une approche générale pour les maillages à faces arbitraires[Khod 02] consiste à coder simultanément les degrés des faces et des sommets. Dans[Lee 02], un parcours à base d'arêtes est associé à un critère géométrique pour la sélection de la bonne arête à franchir lors du passage de la zone visitée à la zone non-visitée.

Codage optimal de graphes

Il est à noter que des progrès dans le codage optimal des graphes planaires ont été réalisés récemment[He 99][Chia 01], basés dans la plupart d'entre eux sur le travail de Schnyder[Schn 90]. Une décomposition particulière de Schnyder d'une triangulation en trois arbres couvrants peut donner lieu à un codage optimal des triangulations en temps linéaire[Poul 03, Poul 06].

2.5.3 Codage de la géométrie

Le schéma classique pour la compression de l'information géométrique associée aux maillages contient trois étapes :

La quantification

Cette étape exploite la limitation de la perception de l'œil humain. Les coordonnées des points sont quantifiées pour passer de la représentation explicite (par exemple une représentation IEEE de flottant sur $32 - bits$) à une précision moins gourmande en bits. Plusieurs versions peuvent être utilisées, allant de la quantification uniforme sur 16 *ou* $8 - bits$, à la quantification non-uniforme[Gers 91]. La quantification vectorielle[Gers 91], a été récemment introduite dans la compression des données associées aux sommets d'un maillage[Chou 02, Lee 00], où le schéma présenté ici est un peu modifié : la prédiction est réalisé en premier, ensuite les résidus d'une position sont codés ensemble par une quantification vectorielle pour exploiter la corrélation existante.

La prédiction des positions

Le but de cette étape est de prédire la position d'un sommet à partir des positions des sommets déjà visités. Plusieurs schémas ont été introduits dans la littérature, comme la prédiction *delta*[Deer 95, Chow 97], où les différences entre les positions des sommets voisins sont gardées au lieu des positions originales. La prédiction linéaire[Taub 98] consiste à remplacer la position actuelle par une combinaison linéaire des sommets déjà visités. La règle du parallélogramme[Toum 98] code l'erreur entre la position actuelle du sommet et celle estimée au quatrième sommet du parallélogramme construit par les trois sommets d'un triangle opposé (voir figure2.14). La prédiction d'ordre deux[Baja 99] est réalisée en deux phase : une prédiction simple, suivie d'une prédiction sur les prédictions.

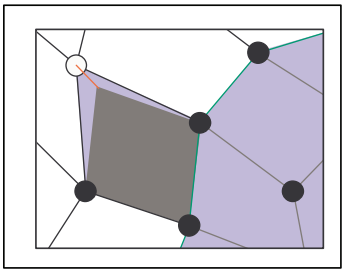

FIG. 2.14: La prédiction de la position de la quatrième position par un parallélogramme (en gris), renvoie une erreur (en rouge) comme Résidu[Isen 00b].

Le codage d'entropie

Le codage entropique[Cove 06], comme le codage de *Huffman*, ou le codage arithmétique ; consiste à compacter le résultat de la compression géométrique associées souvent à la compression topologique, pour construire le bloc comprimé final.

2.6 Les traitements en mémoire auxiliaire

Les représentations utilisées pour les données géométriques (dont les formats commercialisés comme *OBJ* de *Wavefront Technologies* et standards comme *VRML*[ISOIa]) sont souvent indexées : les sommets sont énumérés en premier, avec leurs coordonnées, ensuite les faces sont listées et représentées par les indices de leurs sommets.

Cependant le traitement des maillages gigantesques de l'ordre de milliards de triangles se trouve limité par ces représentations. En effet, à partir d'une face, l'indirection nécessaire pour accéder à ses sommets peut être très coûteuse lorsque le volume à indexer est très large, voire impossible quand ceci dépasse la plage adressable dans la mémoire de la machine. D'où l'intérêt des méthodes suivantes appelées généralement *Out of Core*.

Les techniques de traitement de structures géométriques en mémoire auxiliaire[Cran 06, Silv] sont analogues à celles développées pour minimiser les défauts de pages en mémoire vive[Arge 04]. Et nécessitent souvent un ré-arrangement des données effectué généralement en mémoire externe[Vitt 99].

2.6.1 La soupe de triangles

Une solution intuitive consiste à ne pas énumérer les sommets, et à inclure directement les coordonnées des sommets dans les faces. Cette solution permet de traiter les faces d'un maillage de manière indépendante, et d'éviter l'étape d'indirection. Mais la mise à jour des sommets, et le parcours de son voisinage n'est pas aussi facile que dans le format indexé. Cette technique a été introduite pour plusieurs types d'applications[Lind 00, Lind 01, Wu 03]. La représentation explicite des parties du maillage présentes en mémoire de travail est généralement indexée.

2.6.2 Les structures hiérarchiques

Appelées aussi *les structures multi-résolutions*. Ces structures permettent l'accès au maillage par récurrence sur des subdivisions spatiales. Ces structures sont aussi utilisées pour ajuster le niveau de détails des maillages[Flor 05]. Les structures de base communes entre ces structure sont les arbres (notamment les B-trees[Sedg 84] et ses dérivés[Silv]).

Cignoni et al.[Cign 03] proposent une version adaptée de l'*octree*[Same 90b, Same 90a] dédiée pour les algorithmes génériques *Out of Core* sur les maillages, appelée *Octree Based*

External Memory Mesh. Cette structure repose sur une décomposition du cube englobant la triangulation récursivement jusqu'à ce que la taille désirée des cubes élémentaires (exprimée en nombre de sommets par cube, qui sont indexés localement dans le cube) soit atteinte. Ces cubes élémentaires sont les feuilles de l'arbre, et sont stockées sur le disque externe, et chargées en mémoire à la demande. Une variante de la représentation par *octree* a été adaptée à la multi-résolutions des maillages[Garl 05], dans laquelle les nœuds internes de l'arbre gardent des représentants épars des sommets du niveau inférieur pour permettre une construction grossière à ce niveau là. Dans ces structures, l'occupation mémoire permanente se restreint à l'arbre de recherche des blocs, tandis que les données réelles sont chargées en fonction des besoins.

2.6.3 Les formats *Streaming*

Pour avoir une structure de données adaptée au traitement de maillage, dans le cas où ceci se fait séquentiellement, Isenburg et al.[Isen 05a] ont proposée une structure ordonnée, qu'il appelaient *Streaming Mesh*. Le but est d'avoir un format qui permet de passer le maillage en flot dans la mémoire de travail pour pouvoir le traiter sans avoir besoin de charger des parties résidentes dans la mémoire auxiliaire. Pour ce faire, les sommets et les faces du maillage sont intercalés dans la même structure. Au fur et à mesure que les faces défilent dans la structure, les sommets sont introduits dans le flots (lorsqu'une face les référence), ou finalisés lorsqu'ils ne sont plus référencés par aucune face. Ce format a été utilisé pour la compression des gros maillages[Isen 05b, Isen 03b, Isen 06a], pour la simplification des maillages[Isen 03c], et aussi pour la construction de la triangulation de Delaunay[Isen 06b].

2.7 Les structures de données compactes

Entre les structures de données explicites, et le codage des triangulation, il existe une troisième gamme de structures appelées *structures compactes*. Ce type de structures vise deux objectifs :

- Avoir une structure qui soit exploitable localement : ce qui veut dire que l'accès aux composant se fait en temps constant.
- Minimiser en même temps l'espace occupé par la structure, pour que cette dernière tienne en mémoire de travail, et par conséquent retarder le plus longuement possible le recours au transfert avec la mémoire auxiliaire.

2.7.1 La structure de Blandford et al.

Blandford *et al.*[Blan 03a, Blan 05] ont proposé des structures de données pour les maillages bidimensionnels et tridimensionnels.

La structure de données proposée peut être à base de sommets, ou à base d'arêtes[Blan 06] :

La structure à base d'arêtes

La structure stocke une paire $(cle, donnee)$ pour chaque arête, où la clé est une paire (a, b), et la donnée est la paire (c, d). Cette paire modélise l'arête partagée par les deux faces voisines abc et bad. Si l'une des deux faces n'existe pas (le cas du bord de la triangulation), le sommet de cette face manquante est remplacé par une notation spécifique (0 par exemple). Ce qui revient à coder les liens des arêtes (qui ne sont que deux sommets dans le cas bidimensionnel).

Une seule arête est gardée entre les deux arêtes (a, b) et (b, a), c'est le sommet ayant le point le plus petit qui est toujours en premier par exemple (au sens lexicographique de la comparaison).

Les opérations supportées par cette structure sont :

- $rechercher(a, b)$: rechercher les faces ayant (a, b) comme arête.
- $inserer(a, b, c)$: insérer la face (a, b, c) dans le maillage.
- $supprimer(a, b, c)$: supprimer la face (a, b, c) du maillage.

Le gain en mémoire est obtenu en utilisant les différences d'étiquettes au lieu des étiquettes réelles, ce qui permet de minimiser le nombre de bits nécessaire à la représentation des numéros.

Pour chaque arête, on stocke alors la paire $((a, -a), (c-a, d-a))$. Ensuite, les différences $b-a$, $c-a$ et d_a sont codés avec un code gamma[Elia 75] avec un bit de signe pour indiquer les différences négatives. Les entrées du dictionnaire global sont enfin codées en utilisant un code à longueur variable.

La structure à base de sommets

Cette structure est basée sur le stockage du ***cycle de voisins d'un sommet*** (appelé aussi ***lien***) pour chaque sommet. Le cycle est orienté dans le sens du complexe, et les éléments de ce cycle sont les étiquettes des voisins et non pas les références (ou pointeurs) réelles.

Le dictionnaire global contient des entrées, chacune associée à un sommet. L'entrée du sommet a commence par le code gamma[Elia 75] de $|a|$, le degré de ce sommet.

Les opérations supportées par cette structure sont :

- $rechercher(a, b)$: rechercher les faces ayant (a, b) comme arête.
- $inserer(a, b, c)$: insérer la face (a, b, c) au maillage.
- $supprimer(a, b, c)$: supprimer la face (a, b, c) au maillage.

Comme pour la structure à base d'arêtes, les différences des étiquettes sont utilisées au lieu des étiquettes elles mêmes, et ce pour gagner en mémoire. Le lien du sommet a :

$(a_0, a_1,.. a_n)$ sera représenté par $(a, a_1 - a_0,.. an - a_0)$ (voir la figure 2.15).

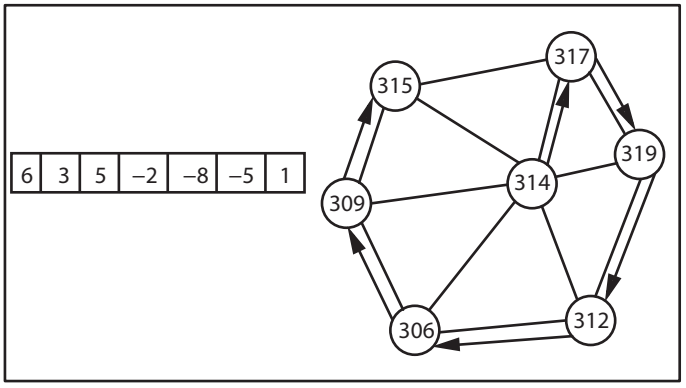

FIG. 2.15: La représentation compacte de Blandford et al. : Le voisinage du sommet 314 est codé par différences par rapport au premier élément de la liste (en l'occurrence 314) ; 6 étant le degré du sommet[Blan 06].

En pratique, le gain dépend de la disposition des sommets dans la structure. La disposition est bonne quand les différences d'étiquettes entre sommets voisins sont minimes. Ce qui permet d'avoir des codes petits pour les différences qui codent les liens des sommets pour la structure à base de sommets, et pour les liens des arêtes pour la structure à base d'arêtes.

Pour réaliser cette distribution, les auteurs se basent sur leurs travaux sur la séparabilité des graphes [Blan 03b, Blan 04]. La procédure consiste en une décomposition récursive en utilisant la médiane de l'axe d'allongement des points (l'axe x ou y dont le diamètre du nuage de point est le plus grand).

Cette procédure suppose que l'ensemble des points est connu à l'avance. Dans le cas contraire, la procédure affecte un étiquetage épars aux sommets, et à chaque insertion, le nouveau sommet obtient une étiquette qui soit proche des ses voisins. Si la plage d'étiquettes est saturée, la procédure ne prévoit aucune solution autre que la reconstruction de la triangulation.

Par conséquent, l'utilisation d'une telle structure pour une construction incrémentale d'une triangulation n'est pas très commode.

2.7.2 La structure de Castelli Aleardi et al.

Dans ses travaux, Castelli Aleardi[Cast 06a] a travaillé sur les représentations compactes des triangulations et des graphes planaires. Sa structure pour les triangulations fera l'objet du travail décrit dans le chapitre 4.

La structure est hiérarchique, et utilise deux niveaux. Le premier niveau est une décomposition de la triangulation en régions de taille $\Theta((\log m)^2)$, où m est le nombre

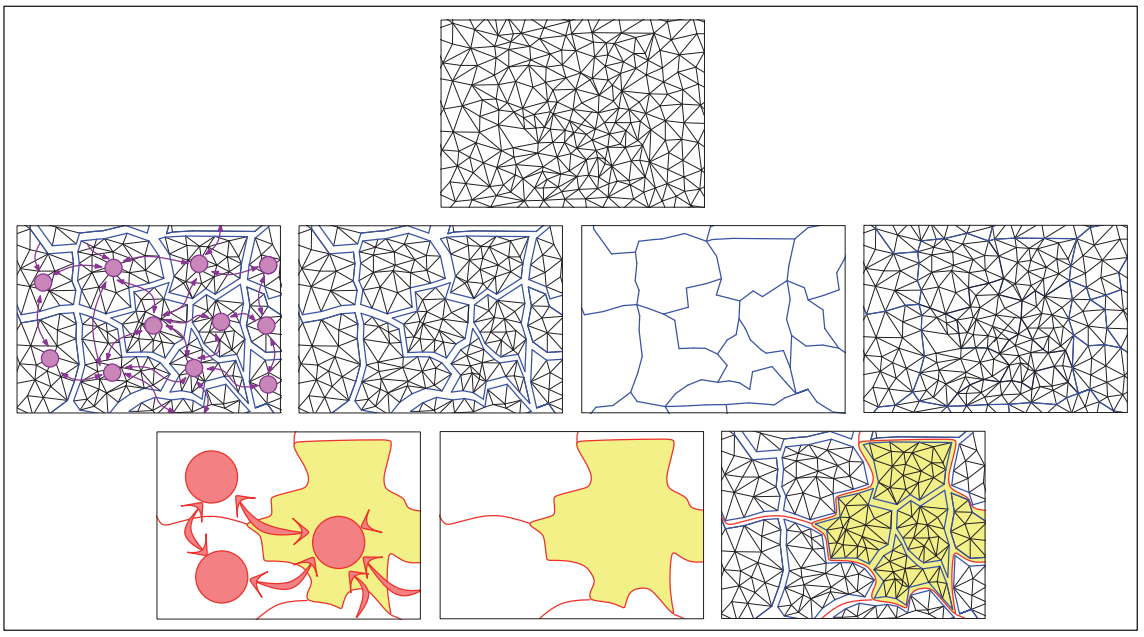

FIG. 2.16: Une représentation en plusieurs niveaux d'une triangulation (de haut en bas, de droite à gauche) : la triangulation est subdivisée en plusieurs micro-triangulations. Ces micro-triangulations sont ensuite regroupées dans des mini-triangulations. Les micro-triangulations sont représentées par des pointeurs vers un catalogue contenant toutes les configurations possibles.

de triangles de la triangulation. Ces régions sont appelées *mini-triangulations*. Ces mini-triangulations sont subdivisées en régions plus petites de taille $\Theta(\log m)$. Ces petites régions sont appelées des *micro-triangulations*.

La structure représente chaque niveau par un graphe : Un graphe pour les micro-triangulations, et un graphe pour les mini-triangulations.

Les micro-triangulations ne sont pas représentées explicitement. La structure garde un catalogue de toutes les triangulations ayant $\Theta(\log m)$ triangles. Chaque micro-triangulation garde uniquement un pointeur vers l'entrée du catalogue ayant la bonne configuration.

Le coût de cette structure est dominé par le coût des pointeurs des micro-triangulations vers le catalogue des micro-triangulations, contenant toutes les triangulations à bord de taille au plus $\Theta(\log m)$. Puisqu'il existe au plus $2^{2,175m}$ triangulations à bord de taille m, le coût de ces pointeurs est $2.175m$ *bits*. Le coût global de cette structure est égal à $2.175m$ *bits* $+ O(m\frac{\log\log m}{\log m})$ (voir figure 4.1).

2.8 Conclusion

Ce chapitre a présenté les grandes lignes des structures de données utilisées pour la représentation des triangulations. Ces structures sont en majorité issues des besoins spécifiques liées aux applications. Ces structures couvrent les représentations explicites où l'accès direct est garanti aux éléments de la structure ; les représentations hiérarchiques, où la structure est subdivisée en blocs, et l'accès se fait au besoin, et ce sont les structures adaptées pour les traitements en mémoire externe et l'ajustement en niveaux de détails. Le codage des triangulation a été aussi présenté, car c'est la solution directe pour gagner en espace mémoire sans se soucier de l'accessibilité aux données. Le reste de la thèse va être consacré aux travaux visant à développer des structures de données pour la représentation des triangulations qui augmentent la capacité de ces structures tout en restant dans les limites de la taille de la mémoire de travail.

Deuxième partie

Structures de données compactes

Chapitre 3

Représentation à base d'indices dans CGAL

3.1 Introduction

La représentation des triangulations dans la bibliothèque *CGAL* utilise des pointeurs comme références vers et entre les entités géométriques manipulées, qui sont les sommets, les faces, et les cellules. Ceci permet une flexibilité vis à vis de l'utilisateur et du programmeur, ainsi qu'une efficacité au niveau de l'exécution. Cependant, pour des petites triangulations (de l'ordre de quelques dizaines de milliers, voir quelques centaines de milliers de sommets), les pointeurs occupant la taille du mot mémoire de la machine s'avèrent un peu dispendieux au niveau de la mémoire allouée.

Dans ce chapitre, les pointeurs sont remplacés par des indices occupant exactement le nombre de bits nécessaire pour la représentation des différentes références des entités de la triangulation. Ce changement implique quelques révisions sur les conteneurs utilisés, et les représentations internes en mémoire des types associés aux simplexes de la triangulation. Cette idée est applicable en dimension 2, comme en dimension 3. Nous avons choisi de l'expérimenter sur les triangulations 3D de CGAL.

Bien que le temps d'exécution soit affecté par cette modification, les gains en mémoire peuvent être très importants, surtout lorsque de nombreuses petites triangulations sont manipulées.

3.1.1 Triangulation vs *Triangulation Data Structure*

La conception des triangulations dans la bibliothèque *CGAL*[CGAL, CGAL 07] sépare les deux aspects de la triangulation, que sont la topologie, et la géométrie (voir section 1.3.1). Cette séparation se traduit au niveau de l'implémentation par l'utilisation de la programmation générique[Alex 01, Vand 02]. En effet, les triangulations sont des

classes instanciées par deux paramètres *template*, l'un pour la topologie, et l'autre pour la géométrie. La définition est de la forme :

```
template < class GT, class TDS_3 > class Triangulation_3;
```

Les instanciations de ces deux paramètres (`GT` et `TDS_3`) doivent satisfaire un certain nombre de règles, ces règles sont rassemblées dans des *concepts*[CGAL 07]. Le *concept* est la définition du type abstrait de la classe utilisée pour instancier le paramètre template correspondant.

Le concept du premier paramètre (`GT`) définit les types de données des objets géométriques tels que le point, le vecteur, et le segment, ainsi que les prédicats nécessaires à la construction de la triangulation tels que le prédicat pour calculer l'orientation de trois points. Le cadre de cette thèse, et de ce chapitre notamment, ne couvre pas cet aspect géométrique, c'est donc le deuxième concept qui nous est intéressant. Le travail concerne alors la topologie, qui est décrite en détail dans la section suivante.

Le deuxième concept est appelé *Triangulation Data Structure*. Dans le reste de ce chapitre, cette appellation est remplacée par son acronyme *TDS_3*.

La validité d'une triangulation en dimension 3 dans *CGAL* est définie par ce qui suit :

- La validité de sa structure de données, qui sera détaillée dans la section suivante.
- L'orientation positive de toutes les cellules de la triangulation.
- Dans le cas dégénéré : chaque deux facettes adjacentes (u, v, w1) et (u, v, w2) ayant l'arête (u, v) en commun, w1 et w2 sont sur deux côtés opposés de (u, v).
- Si tous les points sont colinéaires, pour chaque deux arêtes (u, v) et (u, w), v et w sont à deux côtés opposés de v.

Conventions : Une triangulation de points dans $\mathbb{R}^d$ couvre l'espace $\mathbb{R}^d$ et contient des cellules ayant $d+1$ sommets. Quelques unes sont infinies, et sont obtenues en joignant le sommet infini a toutes les facettes de l'enveloppe convexe des points.

- en dimension 2 : la triangulation ne contient que des points coplanaires.
- en dimension 1 : la triangulation ne contient que des points colinéaires.
- en dimension 0 : la triangulation ne contient qu'un seul point fini (par opposition au sommet infini).
- la dimension -1 est une convention pour désigner la triangulation qui ne contient que le sommet infini.

3.2 La représentation de la TDS_3 de CGAL

La classe `Triangulation_3` de *CGAL*[CGAL 07] est conçue pour représenter la triangulation d'un ensemble de points $\mathcal{A}$ dans $\mathcal{R}^3$. Elle consiste à partitionner l'enveloppe convexe de $\mathcal{A}$ en tétraèdres dont les sommets sont les points de $\mathcal{A}$.

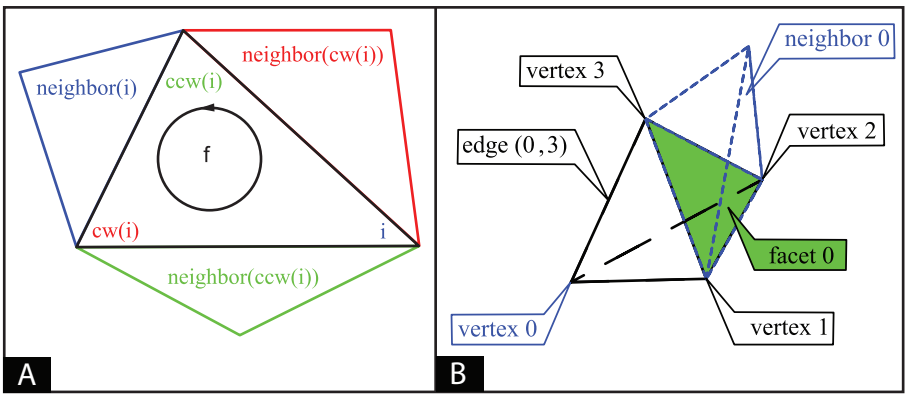

FIG. 3.1: Les conventions des triangulations de *CGAL* en dimension 2 (A) et en dimension 3 (B) : Les sommets sont numérotés dans l'ordre direct, et chaque voisin est opposé au sommet du même indice[CGAL 07]. la fonction $ccw(i)$ retourne le numéro suivant de i dans le sens direct de l'orientation, et la fonction $cw(i)$ retourne le numéro suivant dans le sens opposé.

L'extérieur de l'enveloppe convexe de A est subdivisé en tétraèdres en considérant que chaque triangle sur l'enveloppe convexe est incident à un tétraèdre infini ayant comme quatrième sommet un sommet auxiliaire appelé *sommet infini*. Ce qui permet de garantir que chaque triangle est incident à deux tétraèdres exactement.

La TDS_3 de CGAL représente la triangulation comme un ensemble de sommets (y compris le sommet infini), et de cellules (y compris les cellules infinies), avec des relations d'adjacence :

- Chaque sommet donne accès à une cellule incidente
- Chaque cellule donne accès à ses quatre sommets, et ses quatre voisins.

Les quatre sommets de chaque cellule sont numérotés de 0 a 3 dans une orientation positive définie par l'orientation de l'espace euclidien sous-jacent $\mathbb{R}^3$. Les voisins sont numérotés de telle sorte que chaque voisin d'indice i est opposé au sommet du même indice (voir figure 3.1).

La validité de la TDS_3 est définie par le respect des relations d'adjacence et de voisinage. Ce qui signifie que le voisin déclaré d'une cellule partage bien avec lui les bons sommets.

Facettes et arêtes : Les arêtes et les facettes ne sont pas explicitement représentées dans la triangulation :

- une facette est définie par une cellule et un indice (la facette i de la cellule c, est la facette de c opposée au sommet d'indice i)
- une arête est définie par une cellule c, et deux indices (l'arête (i, j) de la cellule c est l'arête dont les extrémités sont les sommets de c d'indice i et j).

3.2.1 L'implémentation

La TDS_3 est définie en utilisant deux paramètres template, sous cette forme :

```
template < class VB, class CB > class TDS_3;
```

Le premier paramètre `VB` sert pour définir le type des sommets de la triangulation, et le deuxième `CB` sert pour définir le type des cellules. Le diagramme dans la figure 3.2 illustre les dépendances entre les types de la TDS_3, et entre la TDS_3 et la géométrie de la triangulation.

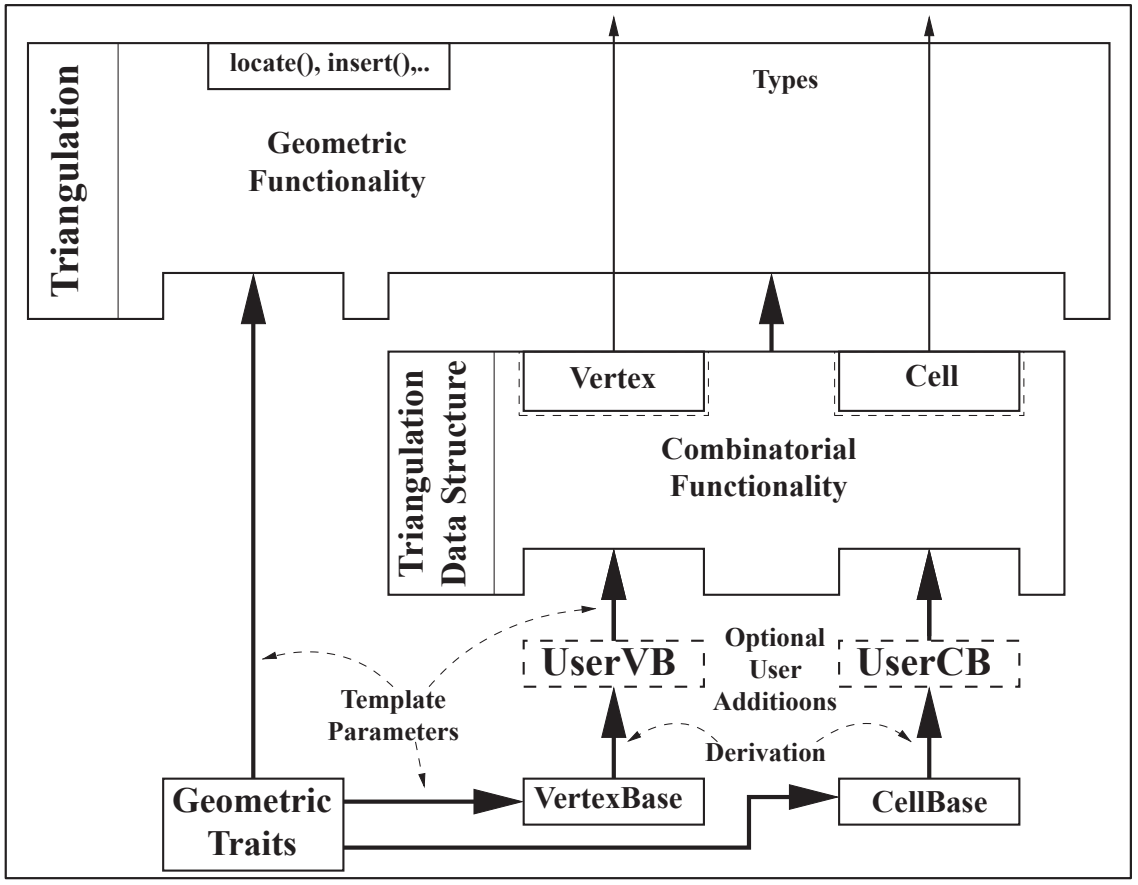

FIG. 3.2: L'architecture de la triangulation de CGAL : la triangulation fournit les méthodes nécessaires pour l'utilisateur comme celle qui insère un nouveau point. Cette triangulation est définie par deux paramètres template : *GeometricTraits* et *TriangulationDataStructure*, le premier fournit les types et méthodes géométriques faisant appel aux coordonnées des points, le deuxième fournit les types et méthodes topologiques. La *TriangulationDataStructure* est définie par deux paramètres template, *UserVb* et *UserCB* qui servent à définir les types des sommets et cellules de la triangulation. Ces types peuvent être définis par l'utilisateur pour inclure davantage d'informations dans les simplexes. L'utilisateur peut aussi dériver ces classes à partir des classes de base *VertexBase* et *CellBase*, et y ajouter ses informations supplémentaires.

Une composante très importante en ce qui nous concerne dans cette classe est le conteneur. Les conteneurs de sommets et de cellules maintiennent l'essentiel de la triangulation puisque on y trouve les objets eux mêmes. Et toutes les opérations d'accès et de mise à jour portent sur ces éléments. C'est donc cette composante qui définit la manière dont la mémoire est utilisée. Pour réduire le coût en mémoire, deux directions sont envisagées :

- Modifier le mode de représentation de l'information, en changeant les définitions des types abstraits de données.
- Modifier les représentations internes de ces objets en mémoire, ainsi que l'implémentation des méthodes d'accès, et des méthodes d'allocation et de restitution de la mémoire.

Cette partie de travail explore la deuxième piste. Et c'est donc la structure du conteneur qui sera modifiée pour qu'elle soit adaptée à des nouvelles représentations internes des données que sont les sommets et les cellules de la triangulation.

3.3 Représentation à base d'indices

L'idée présentée ici consiste à éviter l'utilisation des références absolues de la taille du mot mémoire de la machine (32 ou 64 bits), en utilisant des numéros ou indices comme références sur une taille arbitraire de bits. L'idée en elle même est triviale en termes de codage, mais le passage à la pratique pour des représentations explicites de triangulation nécessite quelques considérations, en particulier :

- Ces indices vont référencer des emplacements mémoire dans des conteneurs différents, ce qui implique le besoin de localisation de ces conteneurs, avec l'ajout d'un niveau d'indirection supplémentaire.
- L'alignement en mémoire est sur 8 bits, d'où la nécessité de recourir à des opérations élémentaires sur les bits pour pouvoir stocker des indices sur des tailles arbitraires de bits.

La représentation interne

Les conteneurs standards[Muss 95, Brey 97, Stan 05] manipulent les données qu'ils contiennent sans se soucier de leurs définitions. Il sont définis de manière générique sous la forme :

```
template < class T, class Allocator = allocator<T> > class vector;
```

L'utilisateur définit le type T de données stockées dans les conteneurs, et peut fournir l'allocateur *Allocator* définissant l'aménagement mémoire : comment l'espace est alloué, et comment il est restitué.

Le *Compact_container* de CGAL[CGAL 07] adapte cette définition à la spécificité des objets de la triangulation. La gestion des espaces de la mémoire, notamment le repérage

des positions libres et occupées dans une plage mémoire allouée pour le conteneur est basée sur une information fournie par l'objet en mémoire lui même[1].

Les types abstraits associés aux données des conteneurs correspondent aux objets de la triangulation : les sommets et les cellules. Ces sommets et cellules ne sont qu'une suite de champs, chacun contenant une référence (de sommet ou de face). Dans la représentation de *CGAL*, ces références sont absolues et représentant directement des adresses mémoire (ou pointeurs) avec lesquelles (sans aucune information supplémentaire) l'objet déréférencé (sommet ou face) peut être localisé directement, et lu ou modifié.

Dans notre travail, la définition des types ne change pas, mais la représentation interne de ces types dans le conteneur change. Les champs des sommets et cellules ne seront plus des références absolues, mais des indices dans un conteneur. Par conséquent, les conteneurs des sommets et des cellules ne sont plus que des suites d'indices sans contexte. Cette suite d'indices est stockée en mémoire sous forme d'une chaîne de bits. Pour accéder à un objet à partir d'un champ donné, l'indice seul ne fournit pas l'information complète. L'information supplémentaire est le contexte permettant le passage de cet indice à l'objet réel.

C'est alors une couche supérieure qui assure l'extraction de l'information d'une simple chaîne de bits et sa modification. La structure du conteneur n'est plus analogue à celles des conteneurs standards.

L'architecture

Le diagramme dans la figure 3.3 montre l'architecture d'une implémentation utilisant l'idée des indices. La classe triangulation maintient la séparation et la concordance entre la géométrie et la topologie représentée par une version modifiée de la TDS_3 appelée $Compact\_TDS\_3$. Une nouvelle séparation est introduite dans cette $Compact\_TDS\_3$, entre les *données* et les *algorithmes*. Pour ce faire, un nouveau concept est défini, et appelé ***Container***. Ce *Container* fournit et définit les types abstraits, les représentations internes en mémoire, la gestion de la mémoire en matière d'allocation et de restitution, et toutes les méthodes d'accès en lecture et en écriture. On y trouve aussi les itérateurs sur ces données. Les définitions sont de la forme :

```
template < class GT, class Compact_TDS_3 > class Triangulation_3;
template < class Container > class Compact_TDS_3;
```

[1] Les références étant des pointeurs, sont alignés sur 4 octets en mémoire, ce qui implique que les deux bits les moins significatifs sont toujours à zéro. Ces deux bits sont utilisés pour cacher l'information *libre* et *occupée* indiquant respectivement si la case en question est libre ou occupée.

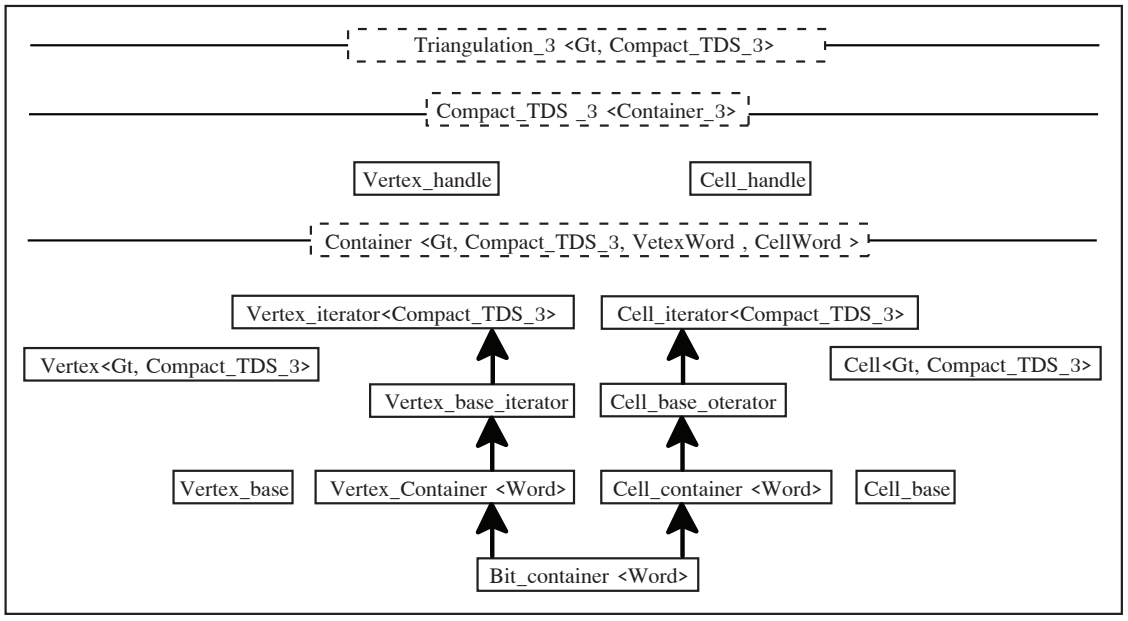

FIG. 3.3: Le diagramme présente les trois couches principales de la triangulation :
(1) La couche supérieure assure l'interface avec l'utilisateur, et est la classe $Triangulation\_3$.
(2) La couche intermédiaire $Compact\_TDS\_3$ fournit les types pour les sommets et les cellules $Vertex$ et $Cell$ à partir de la couche inférieure. Les objets de ces types permettent à l'utilisateur d'accéder à la triangulation, la parcourir, et la modifier. C'est dans ce niveau que les algorithmes de la triangulation sont implémentés.
(3) La couche $Container\_3$ définit la représentation interne des objets de la triangulation : La classe de base est $Bit\_container < word >$, et permet de gérer une suite de bits en mémoire, sous forme d'une suite de valeurs scalaires (*char*, *byte*, ou *int*,..) et d'en extraire des indices à une position donnée, ou d'y écrire aussi. A partir de cette classe, les conteneurs de sommets et de cellules sont dérivés, ainsi que les itérateurs de base permettant de parcourir ces conteneurs de bits. L'information extraite à ce niveau est ensuite formatée pour être utilisée dans la couche supérieure sous la forme de $Vertex\_iterator$ et $Cell\_iterator$.

3.3.1 Les cellules et les sommets

Les types associés aux sommets et cellules dans le `Container_2` de la `Compact_TDS_3` doivent remplir deux fonctionnalités :

- fournir les méthodes d'accès aux champs qui sont la cellule incidente pour un sommet, et les voisins et sommets d'une cellule, en deux modes : lecture et écriture. Pour ce faire, l'utilisateur travaille sur une image de l'élément en mémoire permettant l'accès aux différents champs. Cette image est la variable déclarée dans un programme de haut niveau.
- Garder un référencement pertinent à chaque instant, garantissant la concordance entre l'élément en mémoire, et son image manipulée par l'algorithme. Cet aspect ne se pose pas pour les références absolues, où l'accès aux champs est direct, et la modification d'un champ d'une variable déclarée affecte directement l'emplacement mémoire associée, car l'élément en mémoire est lui même l'image manipulée par l'algorithme. Mais dans le cas des références indices, où l'accès aux éléments en mémoire ne peut se faire directement, la concordance doit être maintenue explicitement. Cette concordance doit être vérifiée avant chaque accès à l'un des champs de la variable, et mise à jour après chaque modification.

Ces deux fonctionnalités sont remplies en définissant les sommets et cellules par deux valeurs :

- Une valeur clé qui est l'indice de l'objet dans son conteneur.
- Une valeur mutable définie comme une image de l'élément réellement gardé en mémoire.

La définition est de cette forme :

```
template < class Compact_TDS_3 > class Cell{
    Compact_TDS_3 * _tds;
    Cell_base * _cb;
    unsigned int _index; ..};
```

La variable mutable peut être omise pour les sommets, vu que la seule valeur qu'ils contiennent est la référence d'une cellule incidente. Cette valeur peut être rendue à la volée, sans variable mutable intermédiaire.

```
template < class Compact_TDS_3 , class GT > class Vertex{
    Compact_TDS_3 * _tds;
    unsigned int _index; ..};
```

La définition de la variable mutable pour les cellules peut prendre cette forme :

```
template < class Compact_TDS_3 > class Cell_base{
    unsigned int * _vertex[4];
    unsigned int * _neighbor[4]; ..}
```

Cette séparation implique la mise à jour de l'image à chaque accès à l'élément, et la mise à jour de l'élément lui même après chaque modification apportée à l'image.

3.3.2 Les itérateurs

Comme les itérateurs standards sont analogues aux références absolues que sont les pointeurs, les itérateurs dans la classe *Container* sont analogues aux nouvelles références que sont les indices. Ces itérateurs ne sont plus liés à des conteneurs, car il n'y a pas réellement de conteneurs, mais sont définis globalement dans la $Compact\_TDS\_3$ comme une surcharge des objets de la triangulation : les sommets et les cellules. Cette surcharge fournit les opérateurs de base de tout itérateur bidirectionnel[ISOIb] qui sont :

- L'opérateur de déréférencement $*$ qui retourne une valeur du type spécifié par la définition de l'itérateur ($Vertex$ pour les sommets et $Cell$ pour les cellules).
- Les opérateurs d'incrémentation et de decrémentation permettant le parcours des éléments stockés en mémoire du type spécifié par la définition de l'itérateur.

Ces itérateurs fournissent l'interface entre les chaînes de bits contenant les sommets cellules, et les algorithmes de la triangulation. De manière générale, l'accès réalise le passage d'un indice i à une valeur t de type T (sommet ou cellule) :

- la position de l'élément dans la suite de bits (les données réelles en mémoire) est calculée à partir de l'indice i. Et la suite de bits b_j représentant l'objet en question t_{brut} est retournée.
- Les champs dans cette suite t_{brut} sont extraits, et sont convertis pour construire la valeur t lisible par la $Compact\_TDS\_3$.

3.3.3 Allocation de la mémoire

Pour des raisons d'efficacité en occupation mémoire, les tableaux de données sont à deux étages (voir figure 3.4). Le premier étage est un tableau de pointeurs vers les tableaux de données. Ces tableaux de données sont alloués suivant le besoin, un par un. La taille de ces tableaux de données est $\sqrt{N}$, où N est le nombre maximal d'éléments. La perte de mémoire c'est à dire l'espace alloué mais non utilisé, est alors au pire $\sqrt{N}$. Ceci dit que la taille maximale de la triangulation doit être connue d'avance.

Bien que cette technique fige la taille des blocs alloués, elle épargne l'utilisation de délimiteurs de début et de fin de bloc dans le cadre d'une allocation dynamique. Sachant que ceci nécessite la connaissance au préalable de la taille maximale de la triangulation. Cette contrainte s'avère raisonnable lors de la manipulation de gros jeux de données. Elle peut quand même être contournée en choisissant une taille arbitraire si celle-ci n'est pas connue par l'utilisateur.

La taille du conteneur est exprimée en nombre d'éléments qu'il contient, c'est à dire le nombre de sommets ou de faces :

- **Le conteneur des sommets** contient les informations stockées dans les sommets : les indices des faces incidentes à ces sommets (voir figure 3.5).

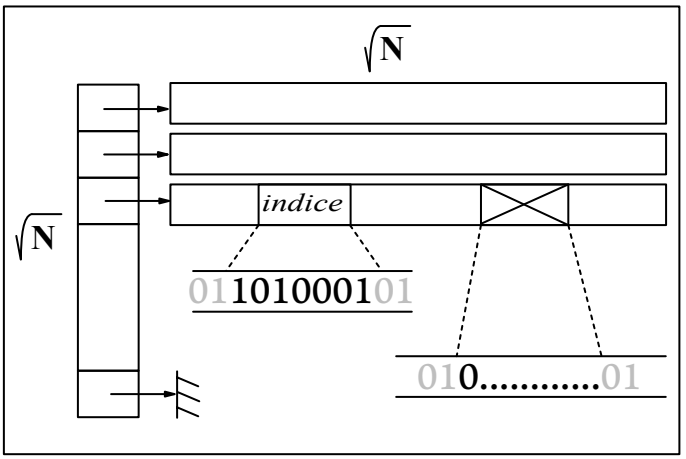

FIG. 3.4: Tableau à deux étages, où les éléments sont loués par blocs de $\sqrt{N}$. Pour repérer les emplacements libres dans un bloc, un bit est réservé dans chaque case pour indiquer sa disponibilité.

- **Le conteneur des cellules** contient les informations stockées dans les cellules : des suites de 8 indices (voir figure 3.5) :
  - 4 indices pour les sommets de la cellule.
  - 4 indices pour les voisins de cette cellule.
- **Les conteneurs des informations supplémentaires** contiennent des éléments géométriques pour les sommets, ainsi que d'autres informations dont le type est défini par l'utilisateur.

Le stockage au bit près n'est fait que pour les deux premiers conteneurs. Ces deux conteneurs sont déclarés comme des blocs d'octets, et la mémoire est allouée aussi par blocs d'octets. Pour les autres conteneurs, les éléments sont alignés sur un nombre entier d'octets, et sont gérés comme des conteneurs standards.

3.4 Analyse et discussion

L'utilisation de $\log n$ bits pour représenter des références parmi n objets est la solution triviale pour un codage compact. Mais en pratique, le recours est toujours aux solutions simples et compatibles avec les langages de haut niveau. L'application de cette solution triviale impose quelques contraintes techniques et logicielles :

- Le seul type de base dans les langages de programmation est le type *byte* ou *char* représenté sur 8 *bits*. D'autres types peuvent êtres utilisés dans le cas ou la taille des indices est supérieure à 8, mais qui sont toujours alignés sur 8 *bits*.
- Les types abstraits ne correspondent pas aux types déclarés dans les couches les plus basse de l'application qui sont les types de base alignés sur 8 *bits*.

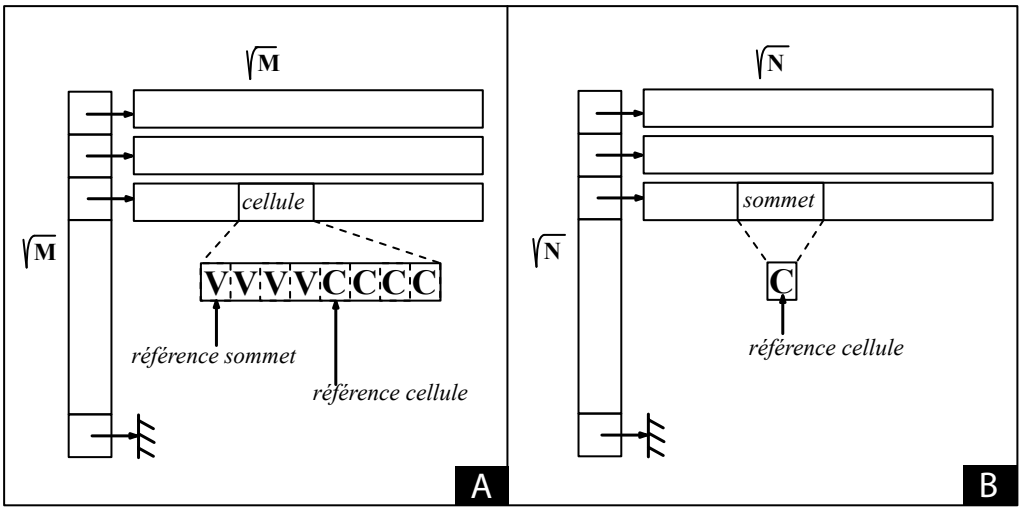

FIG. 3.5: Les conteneurs de sommets et de faces ne sont que des suites de bits, inutilisables en soi.

– La lecture et l'écriture des éléments est indirecte. La suite de bits qui représentent l'information brute est extraite par des opérations de décalage, à chaque accès, qu'il soit en lecture ou en écriture.

3.4.1 Évaluation des coûts

Le gain en mémoire est lié au nombre de bits utilisés pour représenter les indices des cellules et des sommets, et par conséquent au nombre de ces derniers.

Pour une triangulation de n points, produisant m cellules :
– Le coût des pointeurs est de : $(n + 8m)ptr$;
Ce qui donne $(n + 8m) * 32$ sur une machine à 32 bits.
– Le coût des indices est estimé à : $(n)log_2(n) + (8m)log_2(m)$
Pour une triangulation de $2^{20}(\approx 1\ million)$ points, et $2^{24}(\approx 16millions)$ cellules, le coût de la représentation des pointeurs est $(2^{20} + 2^{27}) * 32\ bits$, et celui de la représentation à base d'indices $2^{20}(log(2^{20})) + 2^{27}(log(2^{24}))\ bits$.
Ce qui représente un gain de 25% en espace mémoire. Ce coût est calculé pour la partie topologique uniquement.

Les mêmes données permettent de gagner 62% en mémoire sur une machine à 64 bits. Alors que $10millions$ de points et $160millions$ de cellules permettent de gagner 57% en mémoire.

Il est clair que le gain est lié au nombre de points, ce qui réduit l'efficacité de cette technique pour les très grandes triangulations.

Cependant, cette méthode peut être très efficace pour un grand nombre de petites

triangulations (de quelques dizaines de milliers de points), ou encore pour coder une grande triangulation en plusieurs *sous-triangulations* (par exemple pour des algorithmes out-of-core), où un codage local des entités est utilisé. Ce qui fera l'objet du dernier chapitre de cette thèse.

3.4.2 Résultats expérimentaux

Les résultats suivants présentent la mémoire allouée par différents programmes en fonction du temps d'exécution. Ces programmes se contentent de créer une triangulation de Delaunay en utilisant l'algorithme de la bascule d'arêtes (voir section 1.2.2), et d'y insérer des points dont les coordonnées sont générées aléatoirement suivant une loi uniforme.

Pour avoir une estimation de l'altération du temps d'exécution en utilisant ce type de codage, on compare les temps d'exécution entre les deux versions (la version avec représentation au bit près, et la version utilisant les références absolues) en deux phases : la phase de construction, et la phase de navigation.

Le temps de construction de la triangulation est généralement significatif. Car on accède à tous les objets, et de manière très redondante : le simplexe où le nouveau point sera inséré est d'abord localisé par une marche par visibilité[Devi 01] dans la triangulation, ce qui implique la visite de toutes les cellules sur le chemin. Ensuite, la mise à jour de la triangulation est réalisée après l'insertion du point par des bascules d'arêtes pour rétablir la propriété de Delaunay sur les parties modifiées de la triangulation.

Le temps de navigation est estimé en parcourant la triangulation par les sommets et par les cellules. Ce parcours est réalisé par les itérateurs sur les sommets et les cellules, et donne une très bonne estimation du temps d'accès aux éléments de la triangulation. L'augmentation du temps de calcul n'est pas liée uniquement aux opérations élémentaires supplémentaires pour extraire les données brutes. Mais aussi au nombre d'accès effectués à chaque élément. Car ces opérations de base sont appliquées à chaque accès, qu'il soit en lecture ou en écriture. Et appliquées même en double lors d'une écriture. Ce qui explique le facteur énorme d'augmentation du temps de calcul.

Les valeurs de la mémoire globale dans les tableaux des résultats 3.1 et 3.2 représentent la taille de la structure de données incluant l'espace alloué pour la partie géométrique. Ces valeurs sont les résultats directs des mesures. Les autres valeurs correspondant à l'espace mémoire occupé par la topologie de la triangulation qui est la `Compact_TDS_3` sont calculées à partir des mesures de la mémoire globale.

L'implémentation garde l'interface inchangée par rapport aux triangulations de *CGAL*. Ceci limite davantage l'optimisation en termes de temps d'exécution. En effet, l'accès individuel aux champs des cellules (sommets ou voisins) augmente la redondance lors de l'accès à la mémoire. Ceci peut être considérablement réduit si la lecture peut se faire en blocs, où les opérations de base sur les bits peuvent être optimisées.

La figure 3.6 illustre les résultats détaillés dans les tableaux 3.1 et 3.2.

	nombre de points	mémoire résidente	temps de construction	temps de navigation
pointeurs	100K (673088 cellules)	24,1 Mo TDS — 23,1 Mo	1,7 secondes	0,02 secondes
indices	100K (673088 cellules)	13,8 Mo TDS — 12,8 Mo	1,1 minutes	1,6 secondes
rapport		(58%)	(*40)	(*80)
pointeurs	1M (6749054 cellules)	241,3 Mo TDS — 230,3 Mo	18 secondes	0,2 secondes
indices	1M (6749054 cellules.)	156,3 Mo TDS — 145,3 Mo	11,9 minutes	15,5 secondes
rapport		(65%)	(*40)	(*78)

TAB. 3.1: L'espace mémoire alloué par la structure de données *Triangulation* pour différents jeux de données. Le rapport entre les espaces mémoire est exprimé en pourcentage. On lit aussi le facteur de détérioration du temps d'exécution lors du passage au codage par indices. Ces résultats sont obtenus sur une machine avec un processeur Intel(R) Pentium(R) 4 Core Duo CPU 3.60 GHz avec 2Go de mémoire vive. Les coordonnées des points sont en format *flottant*, représentées sur 4 octets chacune.

3.5 Conclusion

La manipulation de gros jeux de données géométriques nécessite le recours à un codage local des entités. Le but de ce chapitre est de présenter les contraintes techniques lors de l'application de la solution théorique utilisant des indices pour référencer des entités sur le nombre de bits minimal au bit prés.

Le surcoût en termes de temps d'exécution est le premier préjudice de cette méthode, à cause de la phase d'extraction des données brutes de la mémoire. Ce qui peut être extrêmement coûteux par rapport à l'accès direct par des pointeurs.

Toutefois, le retardement du transfert entre la mémoire et les périphériques de stockage est aussi d'un grand intérêt. Les algorithmes à mémoires externes, et les algorithmes hiérarchiques tentent toujours de diminuer les commutations des blocs de travail. L'utilisation d'un codage relatif local avec un codage au bit prés est une solution justifiée qui renforce ce but.

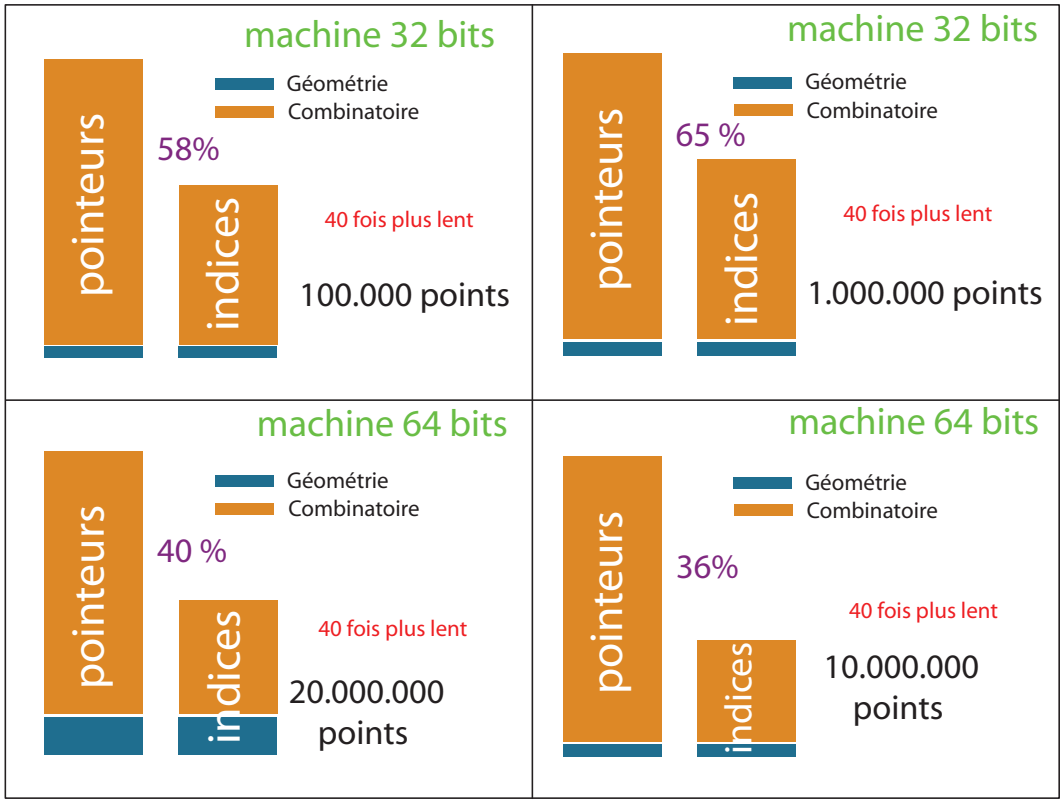

FIG. 3.6: Représentation graphique des gains apportés par l'utilisation des indices comme références dans les triangulations. Les rectangles représentes l'espace mémoire alloué par les différentes structures.

	nombre de points	mémoire résidente	temps de construction.	temps de navigation
pointeurs	10M	4,7 Go	2 minutes	3 secondes
	(67592369 cellules)	TDS — 4,5 Go		
indices	10M	1,8 Go	1,25 heures	1,7 minutes
	(67592369 cellules)	TDS — 1,6 Go		
rapport		(36%)	(*38)	(*34)
pointeurs	20M	9,3 Go	4 minutes	6 secondes
	(135216730 cellules)	TDS — 8,1 Go		
indices	20M	3,7 Go	2,6 heures	3,03 minutes
	(135216730 cellules)	TDS — 3,5 Mo		
rapport		(40%)	(*39)	(*30)

TAB. 3.2: L'espace mémoire alloué par la structure de données *Triangulation* pour différents jeux de données. Le rapport entre les espaces mémoire est exprimé en pourcentage. On lit aussi le facteur de détérioration du temps d'exécution lors du passage au codage par indices. Ces résultats sont obtenus sur une machine 64 bits Bi-Processor Intel(R) Xeon(R) Quad Core CPU 2.33 GHz avec 16 Go de mémoire vive. Les coordonnées sont en format *double* représentées sur 8 octets chacune.

Chapitre 4

Codage de triangulations par catalogues

4.1 Introduction

Des travaux récents entrepris par Castelli Aleardi et al[Cast 04, Cast 05, Cast 06c] ont permis l'élaboration d'une représentation optimale pour les triangulations de la sphère à n points utilisant uniquement $3,24$ n bits par sommet, avec un coût supplémentaire asymptotique négligeable (dans le cas des triangulations à bord polygonal de taille arbitraire, le coût global de la représentation est de $2,17$ bits par triangle).

L'idée de base est le rassemblement des triangles dans des *micro-triangulations* de tailles comprises entre $\frac{\log n}{12}$ et $\frac{\log n}{4}$, et de définir un graphe d'incidence entre ces micro-triangulations.

Une micro-triangulation n'est pas représentée explicitement, mais par un lien vers le bon élément dans une table contenant toutes les configurations possibles de tailles inférieures ou égales à $\frac{\log n}{4}$. La somme des tailles des références vers le catalogue représente le terme dominant exprimé par $3,24$ *bps*, alors que le graphe occupe un espace à coût négligeable (voir figure 4.1).

Toutefois, le terme asymptotique est de la forme $O\left(n\frac{\log\log n}{\log n}\right)$ avec une constante qui fait que ce terme n'est négligeable que pour des valeurs de n de l'ordre de dizaines de milliards. Ce qui rend cette approche purement théorique.

Néanmoins, l'idée en elle même est très intéressante, et peut être appliquée en y apportant quelques simplifications [Cast 06b, Cast 07].

Ce chapitre, présente quelques résultats *non asymptotiques* basés sur les constatations suivantes :

– bien que le coût du graphe d'incidence ne soit pas négligeable, il est toujours plus économique que le graphe original entre les triangles. Et ce, parce que les tailles des

références entre les morceaux de triangulations sont de toute façon plus petites que les tailles des références explicites entre les triangles.

– $\frac{\log n}{12}$ est en fait un petit nombre ($\frac{(\log 70\ \mathit{millliards})}{12} \approx 4$), ce qui justifie l'étude de petits catalogues de taille fixe, pour pouvoir évaluer l'impact sur l'espace mémoire nécessaire à la représentation de la triangulation.

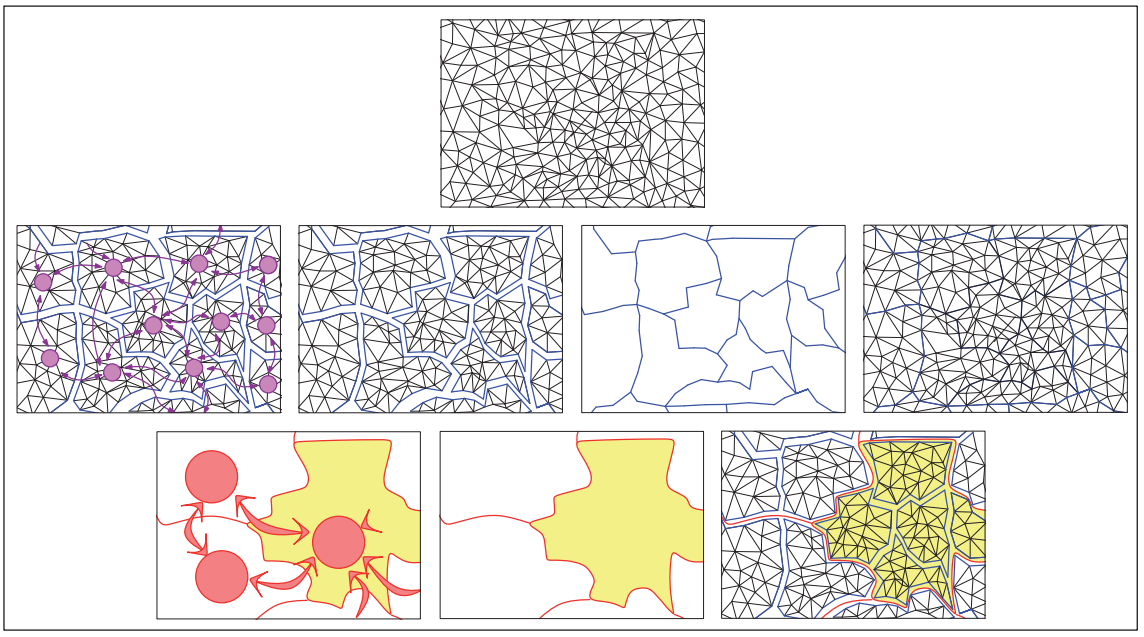

FIG. 4.1: Une représentation en plusieurs niveaux d'une triangulation (de haut en bas, de droite à gauche) : la triangulation est subdivisée en plusieurs micro-triangulations. Ces micro-triangulations sont ensuite regroupées dans des mini-triangulations.

La suite de ce chapitre sera dédiée à l'évaluation en détail de l'espace mémoire indispensable à la représentation des triangulations en dimension 2 en utilisant l'idée des catalogues. Des résultats expérimentaux sont présentés afin d'évaluer en pratique le gain potentiel.

La structure est à base de triangles, et l'interface de la conception fournit les fonctionnalités suivantes :

– Le déréférencement unique de toute face de la triangulation à partir d'une référence, et de tout sommet de la triangulation à partir d'une référence.
– L'accès aux trois sommets ainsi qu'aux trois voisins de toute face de la triangulation.
– L'accès à une face incidente pour tout sommet de la triangulation.
– Le parcours de tous les sommets, ainsi que toutes les faces de la triangulation.

Les structures de données de Castelli Aleardi et al [Cast 04, Cast 05, Cast 06c] utilisent deux niveaux : le niveau des micro-triangulations, et le niveau des mini-triangulations. Les mini-triangulations regroupent plusieurs micro-triangulations et sont de tailles moyennes. Ce deuxième niveau fera la base des structures de données étudiées dans le dernier chapitre

de cette thèse (voir le chapitre 5).

4.2 Définitions

Un ***Catalogue*** $\mathcal{C}$ est une collection $\{t_1, \ldots, t_p\}$ de triangulations planaires à bords polygonaux simples de tailles arbitraires, appelées *micro-triangulations*.

Un ***catalogue stable*** pour une opération de mise à jour donnée est un catalogue tel que le résultat de toute application de cette opération sur une triangulation composée de micro-triangulations de ce catalogue (la partie modifiée après l'application de l'opération), est soit une micro-triangulation du catalogue, ou décomposable (en utilisant éventuellement une partie de son voisinage[1]) en micro-triangulations du même catalogue. Des exemples de catalogues stables sont donnés dans la figure 4.2.

L'intérêt d'un tel catalogue est que n'importe quelle triangulation $\mathcal{T}$ peut être décomposée en micro-triangulations de ce catalogue, et construite incrémentalement en n'utilisant que des micro-triangulations de ce catalogue $\mathcal{C}$: $\mathcal{T} = \bigcup_{j \in J} t_{j_i}$ $(1 \leq j_i \leq p)$.

Un catalogue $\mathcal{C}$ est ***minimal*** s'il ne contient aucune micro-triangulation t_i qui peut être obtenue comme union disjointe d'autres micro-triangulations du catalogue $\mathcal{C}$.

Les opérations de base dans ce chapitre sont l'insertion d'un nouveau point dans la triangulation, la suppression d'un sommet de degré trois, et la bascule d'arête entre deux triangles.

Une triangulation représentée en utilisant un catalogue $\mathcal{C}$ est alors décomposée en sous triangulations, où chaque sous triangulation correspond à un élément du catalogue. Il n'y a alors plus de représentation explicite pour les triangles que si le catalogue en question $\mathcal{C}$ contient le triangle unique comme micro-triangulation.

4.2.1 Construction d'un catalogue

Les catalogues C_i (i est le nombre minimal de triangles par micro-triangulation) étudiés dans ce chapitres ont été construits en fixant le nombre minimal k de triangles par micro-triangulation. Et le résultat s'obtient manuellement de la manière suivante :

- La première étape consiste à insérer toutes les micro-triangulations à k triangles dans le catalogue.
- Les micro-triangulations ayant entre $k+1$ et $2k-1$ triangles doivent aussi figurer dans le catalogue. Car, il n'y a aucun moyen de les représenter par des micro-triangulations de k triangles.
- La dernière étape consiste à explorer toutes les configurations résultant de l'application des opérations de mise à jour sur les éléments déjà insérés dans le catalogue,

[1] Pour les catalogues étudiés dans ce chapitre, la décomposition se restreint aux voisins directs.

et ce jusqu'à ce que tous les résultats soient dans le catalogue ou décomposables en éléments du catalogue.

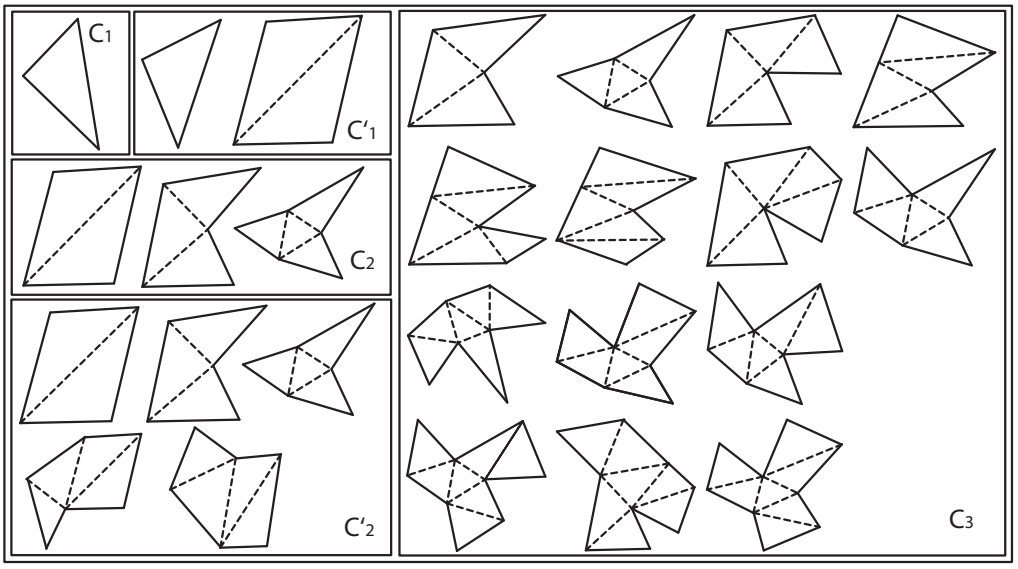

FIG. 4.2: Cinq catalogues stables pour les opérations de base (insertion, suppression, et bascule d'arêtes) : (1) Le catalogue trivial $\mathcal{C}_1$ ne contenant qu'une micro-triangulation à triangle unique (2) Le catalogue Triangle-Quadrangle $\mathcal{C}'_1$ (3) Le catalogue minimal $\mathcal{C}_2$ contenant au moins 2 triangles par micro-triangulation (4) Un catalogue non minimal $\mathcal{C}'_2$ à au moins 2 triangles par micro-triangulation (5) Le catalogue minimal $\mathcal{C}_3$ à au moins 3 triangles par micro-triangulation.

4.3 Catalogues simples

Il est intéressant de voir maintenant l'intérêt que peuvent apporter les catalogues à la représentation des triangulations en termes d'espace mémoire. Pour ce faire, il est souhaitable de calculer quelques bornes supérieures sur la mémoire requise par cette structure.

Le premier catalogue trivial est celui contenant un triangle unique. Une triangulation de n points peut être codée en utilisant $13n$ références.

La conception d'une telle structure est directe et simple. L'indexation est directe, chaque référence $face$ déréférence un objet unique dans le tableau des triangles. Un second tableau pour les sommets est nécessaire.

4.3.1 Le catalogue Triangle-Quadrangle

Le niveau directement supérieure à la représentation à base de triangles explicite s'obtient en ajoutant la micro-triangulation quadrangle au catalogue.

Il est évident que ce catalogue $\mathcal{C}'_1$ est stable pour les trois opérations considérées, mais il n'est pas minimal.

Une triangulation représentée par ce catalogue est un ensemble de trois tableaux : un pour les sommets de la triangulation, un deuxième pour les triangles, et un troisième pour les quadrangles.

La représentation des triangles reste inchangée :

– trois références vers les sommets du triangle.

– trois références vers les voisins de ce triangle.

La représentation des quadrangles est la suivante :

– quatre références vers les sommets du quadrangle.

– quatre références vers les voisins de ce quadrangle.

Ce qui nous permet de gagner 4 références pour chaque quadrangle (8 références au lieu de 12 dans le cas où les deux triangles sont représentés séparément).

Le codage des diagonales des quadrangles se fait implicitement en fixant un ordre conventionnel des sommets. L'ordre choisi est de fixer la diagonale entre les sommets 1 et 3, voir la figure 4.3 gauche.

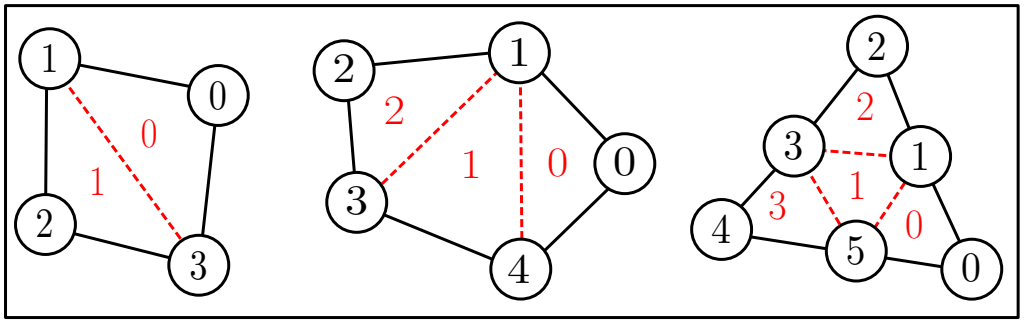

FIG. 4.3: Les diagonales des petites micro-triangulations peuvent être codées implicitement en fixant un ordre conventionnel relatif à la diagonale. La diagonale du quadrangle relie les sommets 1 et 3. Celles du pentagone relient les sommets 1 à 3, et 1 à 4. Et les diagonales de l'hexagone relient les sommets 1, 3 et 5. Les triangles internes peuvent aussi être implicitement numérotés sans ambiguïté.

Le gain en espace mémoire est proportionnel au nombre de quadrangles construits. Ce nombre ne peut pas dépasser n, induisant un coût global de $9n$ références pour une triangulation de n points (le cas d'une quadrangulation). Cependant, il n'est pas toujours possible, ou facile de construire une quadrangulation à partir d'une triangulation.

Construction statique de quadrangulations

Dans le mode statique, la triangulation est entièrement construite, et le travail consiste en la décomposition de cette triangulation en micro-triangulations du catalogue (triangles

et quadrangles).

Plusieurs approches ont été proposées pour la conversion de triangulations en quadrangulations [Heig 83][Rama 98]. Ou pour créer des quadrangulations à partir d'un ensemble de points[Tous 95].

Il n'existe pas une méthode rigoureuse pour réaliser cette conversion, et c'est même impossible dans certains cas (lorsque la taille du bord est impaire par exemple[Bose 97]). Toutefois, il est toujours possible de convertir une triangulation d'une sphère topologique (une triangulation sans bord) en une quadrangulation. Ceci est directement déductible du théorème de Petersen[Pete 91]. Le théorème garantit pour chaque graphe régulier de degré 3, sans isthme, l'existence d'un appariement complet. Ces conditions étant satisfaites par le graphe dual d'une triangulation d'une sphère topologique, il est alors, toujours possible de regrouper tous les triangles en quadrangles deux à deux.

Construction dynamique de quadrangulations

Dans le mode dynamique (ou incrémental), les opérations de mise à jour sont prises en compte, vu que la triangulation est construite et modifiée à la demande. On doit pouvoir insérer de nouveaux points, supprimer des sommets de degré 3, et basculer des arêtes.

Pour réaliser ces opérations en temps constant, la modification doit affecter une partie de taille fixe de la triangulation (voir juste le voisinage direct du triangle impliqué dans la mise à jour).

Cette contrainte nous empêche de profiter du résultat dans le cas statique, et l'utilisation uniquement des quadrangles pour représenter une triangulation n'est plus possible, même pour les triangulations sans bord. La figure 4.4 illustre les cas de conflit pour les opérations de mise à jour, où on ne peut se restreindre à des quadrangles.

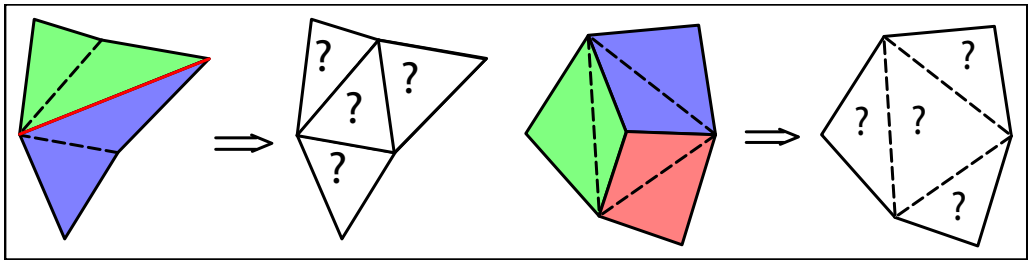

FIG. 4.4: Une triangulation en mode dynamique ne peut pas être représentée que par des quadrangles. La bascule d'arêtes et la suppression d'un sommet de degré 3 peuvent produire une configuration qui ne peut être décomposée en quadrangles (les cas d'un triangle entouré de trois triangles voisins).

Lemme Soit une triangulation planaire $\mathcal{T}$ à n sommets. Une représentation à base de triangles, utilisant le catalogue $\mathcal{C}'_1$ et nécessitant $10,6n$ références se maintient en mode

dynamique.

Démonstration On suppose une décomposition de la triangulation $\mathcal{T}$ en utilisant le catalogue $\mathcal{C}'_1$ et ne contenant pas de triangles adjacents. Le maintien d'une telle décomposition est trivial, et se réalise en parcourant après chaque application d'une opération le voisinage du triangle affecté, et le regroupant avec un voisin éventuel pour former un nouveau quadrangle.

Si t et q représentent respectivement le nombre de triangles et de quadrangles dans la décomposition de $\mathcal{T}$, et b le nombre d'arêtes du bord de la triangulation. Le nombre total de triangles est donné par la formule :

$$2n - b - 2 = t + 2q; \tag{4.1}$$

Le nombre d'arêtes est $3n - b - 3$, et est égal à :

$$3n - b - 3 = a_{int} + a_{tr/quad} + a_{quad/quad} \tag{4.2}$$

où a_{int} est le nombre d'arêtes internes aux quadrangles (le nombre de diagonales), $a_{tr/quad}$ est le nombre d'arêtes partagées entre triangles et quadrangles, et $a_{quad/quad}$ est le nombre d'arêtes entre quadrangles ; puisque la décomposition ne contient pas de triangles adjacents, il n'y a pas d'autres types d'arêtes.
Sachant que

$$a_{int} = q$$

et comme on n'a pas deux triangles adjacents[2]

$$a_{tr/quad} = 3t$$

On arrive à

$$a_{quad/quad} = 3n - 3t - q - b - 3 \geq 0 \tag{4.3}$$

A partir de 4.1 et 4.3, on obtient :

$$q > \frac{3}{5}n - \frac{2}{5}b + \frac{3}{5} \tag{4.4}$$

Le nombre de triangles dans la décomposition est alors au maximum $\frac{4}{5}n - \frac{1}{5}b - \frac{6}{5}$. Le coût global est donc borné par $\frac{53}{5}n = 10.6n$ au lieu de $13n$ références dans la représentation de base, ce qui représente un gain d'au moins 19%.

[2]On considère qu'il n'y a pas de triangles sur le bord de la triangulation pour simplifier les calculs.

La taille des références

La représentation à base de triangles (utilisant le catalogue trivial C_1) utilise $\log n$ bits pour les références des n sommets, et $1 + \log n$ bits pour les références des $2n$ faces (triangles). Pour les relations de voisinage, il est possible d'utiliser un indice associé à chaque référence dans $(\{0, 1, 2\})$ pour coder facilement la référence réciproque (l'indice de la face courante dans la face voisine). Ce qui induit une taille de $3+\log n$ bits par référence face.

Pour une triangulation représentée par le catalogue Triangle-Quadrangle ($\mathcal{C}'_1$), le nombre de quadrangles et le nombre de triangles sont inférieurs à n. Ce qui nous permet de coder les références des faces de la triangulation sur $\log n$ avec 3 bits en supplément : un bit pour différencier les quadrangles des triangles, et deux bits pour la référence réciproque comme avant.

La taille de la référence est alors la même que pour le catalogue trivial. Ce qui ne représente aucun surcoût en termes de taille des références.

4.3.2 Catalogues avec au moins deux triangles par micro-triangulation

Le catalogue minimal

Considérons en premier le catalogue minimal $\mathcal{C}_2$ avec au moins 2 triangles par micro-triangulation (voir la figure 4.2). Ce catalogue contient le quadrangle, le pentagone, et un hexagone.

La représentation des quadrangles ne change pas. Les pentagones sont représentés par 10 références, 5 pour les sommets et 5 pour les voisins. Tandis que les hexagones utilisent 12 références, 6 pour les sommets, et 6 pour les voisins. Ceci économise 8 et 12 références pour respectivement, les pentagones et les hexagones (10 références au lieu de 18, et 12 au lieu de 24 si les triangles sont représentés séparément).

Pour ne pas compliquer la mise en œuvre, on tolère l'auto-voisinage des pentagones et hexagones (voir la figure 4.5), où ces micro-triangulations peuvent être voisins à eux mêmes. Cette faculté mène à des micro-triangulations ayant des sommets à double occurrence, mais facilite énormément la mise en œuvre.

La stabilité Ce catalogue est stable pour les opérations de mise à jour : insertion d'un nouveau point, suppression d'un sommet de degré 3, et bascule d'arêtes. Les figures 4.6, 4.7, et 4.8 illustrent la stabilité pour ces trois opérations.

Il est clair que le nombre maximal de quadrangles que la décomposition d'une triangulation peut avoir est égal à n, ce qui permet d'épargner deux pointeurs sur chaque triangle, passant de $13n$ référence à $9n$ références, induisant un gain d'au moins 13% sur l'espace mémoire. Mais on peut encore garantir mieux.

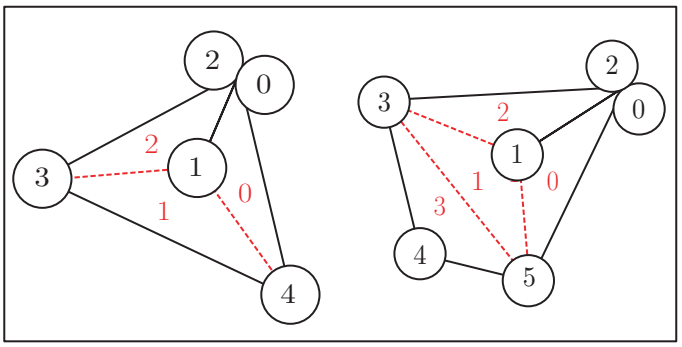

FIG. 4.5: Les pentagones et les hexagones du catalogue minimal $\mathcal{C}_2$ peuvent avoir des sommets internes, et ce en étant des voisins à eux mêmes. Ceci induit la coïncidence des sommets 0 et 2 pour les pentagones, et de deux sommets pour les hexagones (0 et 2, 2 et 4, ou 4 et 6).

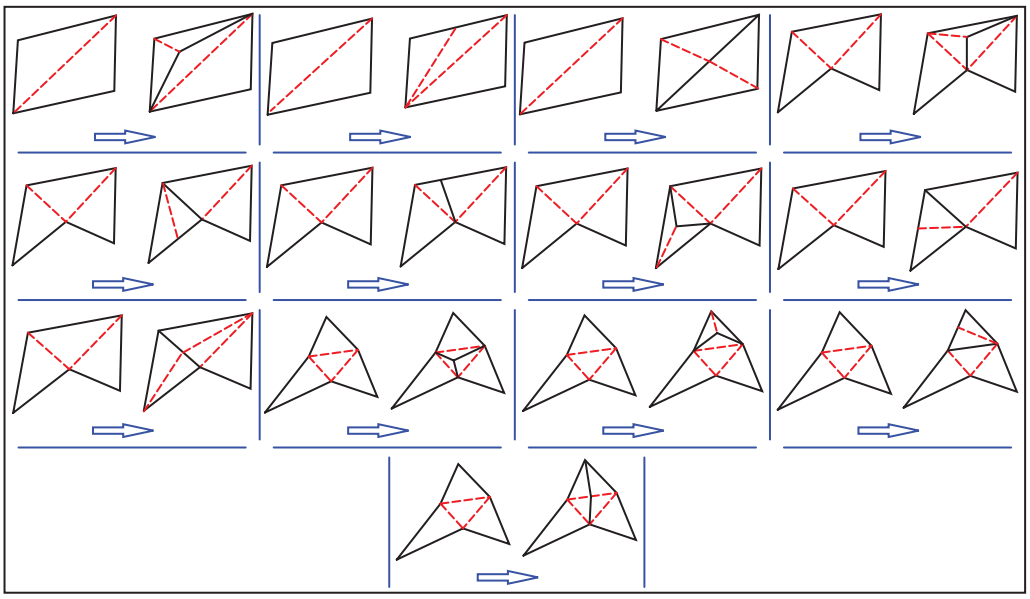

FIG. 4.6: Le catalogue $\mathcal{C}_2$ est stable pour l'insertion d'un nouveau point : l'insertion dans un triangle, ou sur une arête produit toujours une configuration décomposable en micro-triangulations du catalogue $\mathcal{C}_2$; L'insertion en dehors de l'enveloppe convexe de l'ensemble des points construit des nouveaux triangles connexes qui peuvent toujours être décomposés en micro-triangulations, comme par exemple les regrouper deux à deux en quadrangles, et regrouper les trois derniers en pentagone. Cette illustration définit un schéma qui est utilisé pour mettre à jour la triangulation après chaque insertion d'un nouveau point.

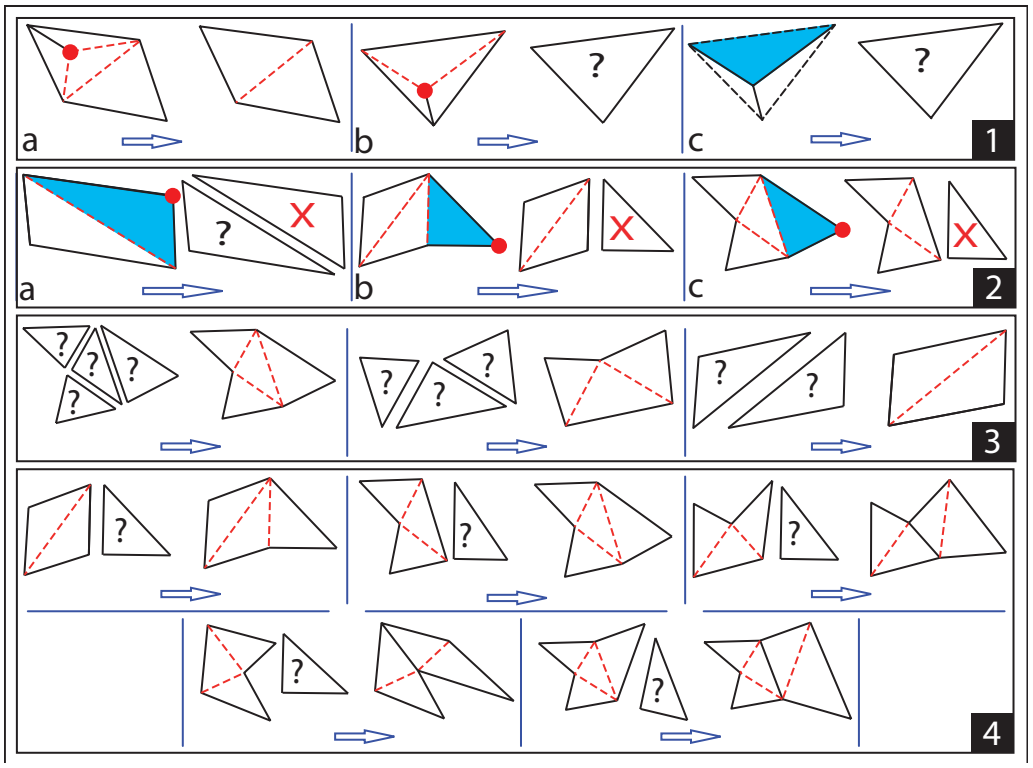

FIG. 4.7: La stabilité du catalogue $\mathcal{C}_2$ pour la suppression d'un sommet de degré 3.
(1.a et 1.b) La suppression d'un sommet interne à un pentagone ou un hexagone (les micro-triangulations acceptant un auto-voisinage) produit une micro-triangulation du catalogue (un quadrangle) ou un triangle isolé.
(1.c) La suppression d'un sommet partagés par deux ou trois micro-triangulations produit un triangle au milieu (celui qui contenait le sommet supprimé).
(2) Une micro-triangulation incidente au sommet à supprimer peut se trouver après la suppression avec un seul triangle (-a- le cas d'un quadrangle), deux triangles (-c- le cas d'un pentagone), ou trois triangles (-d- le cas d'un hexagone). Le pentagone se transforme en un quadrangle, et l'hexagone se transforme en un pentagone ; le quadrangle se transforme en un triangle isolé qui doit être regroupé avec un autre triangle isolé ou intégré dans une micro-triangulation voisine pour construire une nouvelle micro-triangulation du catalogue. On peut avoir deux ou trois micro-triangulations incidentes au sommet à supprimer ; après la suppression, chaque sous-triangulation produira au maximum un triangle isolé (le cas d'un quadrangle) après la suppression. Avec le triangle généré au milieu, on aura un, deux, trois, ou quatre triangles isolés générés après une opération de suppression.
(3) Le regroupement des triangles isolés et l'adjonction d'un seul triangle isolé sont directs. La décomposition déduite est toujours fidèle au catalogue $\mathcal{C}_2$.

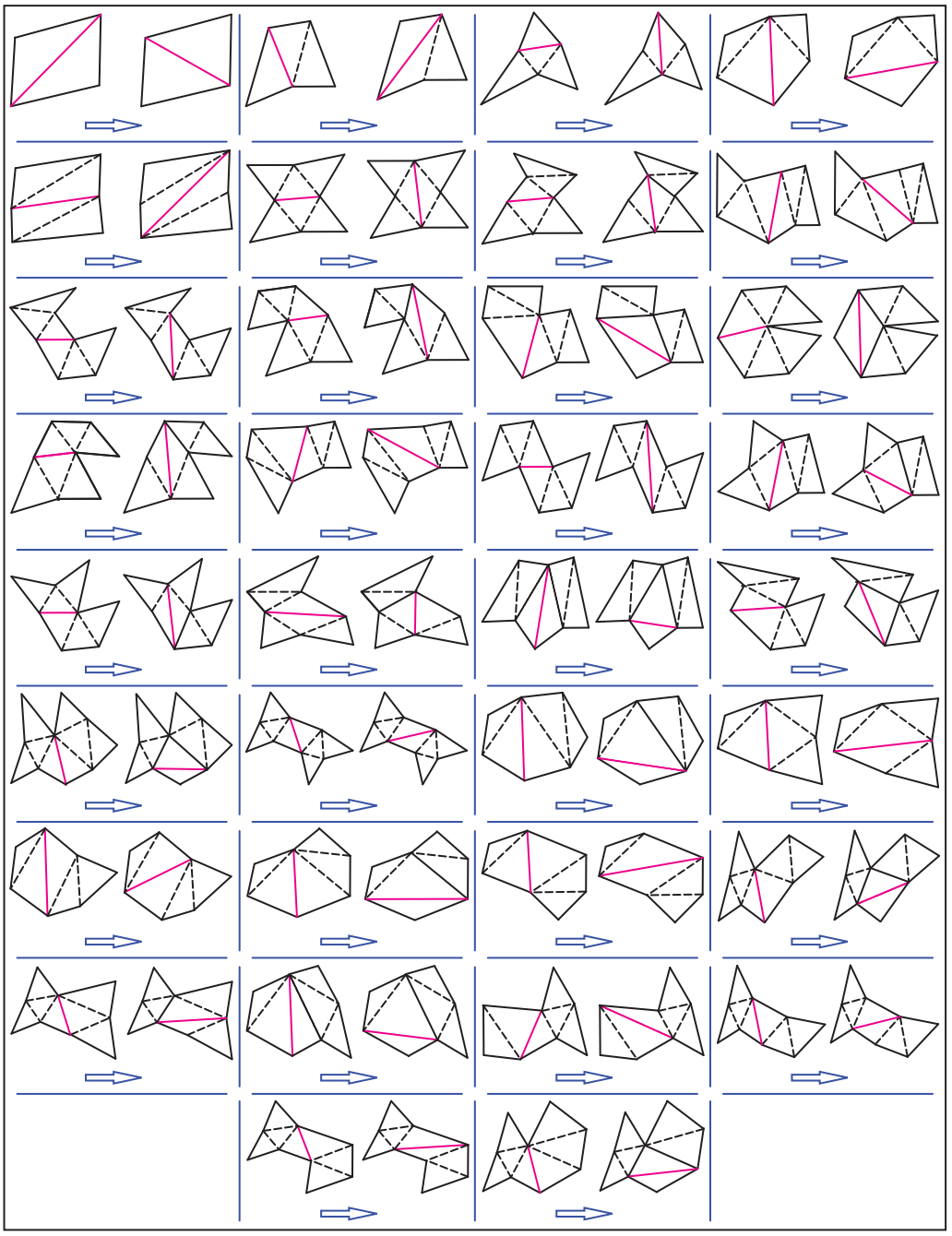

FIG. 4.8: Les cas de figure pour le résultat de l'application de l'opération de bascule d'arêtes aux micro-triangulations du catalogue $\mathcal{C}_2$. Cette illustration définit un schéma qui est utilisé pour mettre à jour la triangulation après chaque bascule d'arête.

Lemme Soit $\mathcal{T}$ une triangulation planaire à n sommets. Une représentation à base de triangles, utilisant le catalogue C_2 et nécessitant seulement $8.5n$ références peut être maintenue dynamiquement.

Démonstration Considérons une décomposition de la triangulation $\mathcal{T}$ ou chaque quadrangle n'est pas voisin à plus de deux autres quadrangles.

Ceci est faisable, car si un quadrangle a trois voisins qui sont des quadrangles, deux d'entre eux sont incidents à un sommet touché par la diagonale du quadrangle, et par conséquent, l'ensemble de ces deux quadrangles et le quadrangle courant peuvent être regroupés en deux pentagones (voir la figure 4.9).

Cette propriété se maintient, en parcourant après chaque opération de mise à jour le voisinage des nouveaux quadrangles créés pour s'assurer que le nombre de quadrangles voisins est bien inférieur à trois, sinon, on regroupe les quadrangles adjacents.

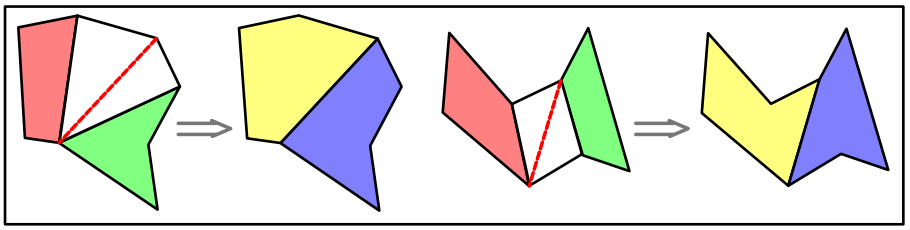

FIG. 4.9: Si deux quadrangles sont voisins à un autre quadrangle, et si ces deux quadrangles sont incidents à un sommet de la diagonale, les trois quadrangles peuvent être convertis en deux pentagones.

Soit q, p et h les nombre de quadrangles, de pentagones, et d'hexagones respectivement dans la décomposition de $\mathcal{T}$. Et soit b le nombre d'arêtes du bord. Pour avoir le cas le pire en termes d'espace mémoire utile, on suppose qu'il n'y a pas d'hexagones dans la triangulation. Le nombre total de triangles est :

$$2n - b - 2 = 2q + 3p \tag{4.5}$$

Le nombre d'arêtes de la triangulation est :

$$3n - b - 3 = a_{int} + a_{quad/quad} + a_{rest} \tag{4.6}$$

Où $a_{int} = 2p + q$ est le nombre d'arêtes internes aux quadrangles et aux pentagones (les diagonales) ; et a_{rest} est la somme des nombres d'arêtes partagées entre pentagones, et entre pentagones et quadrangles.

Et puisqu'il n'y a pas plus que deux quadrangles adjacents à chaque quadrangle, $a_{quad/quad}$ est borné par $2q/2$, et par conséquent a_{rest} est au moins $2q$, ce qui donne :

$$a_{rest} = 3n - 2q - 2p - b - 3 \geq 2q \tag{4.7}$$

Utilisant 4.5 et 4.7, nous avons :

$$p > \frac{1}{4}n - \frac{1}{4}b - \frac{1}{4} \tag{4.8}$$

Le nombre de quadrangles est donc au plus $\frac{5}{8}n - \frac{7}{8}b - \frac{5}{8}$. La triangulation peut être représentée avec $\frac{17}{2}n = 8.5n$ références, au lieu de $13n$ dans la représentation basique, avec un gain d'au moins 35%.

Un autre catalogue non minimal

Une alternative consiste à utiliser le catalogue non minimal $\mathcal{C}'_2$ décrit dans la figure 4.2. Ce catalogue incorpore en plus des micro-triangulations du catalogue précédent (voir le paragraphe 4.3.2) les configurations restantes de l'hexagone. Ce catalogue permet d'imposer en plus que deux quadrangles ne soient jamais voisins, vu que chaque paire de quadrangles voisins peut être convertie en un hexagone. Cette hypothèse, avec la même approche de calcul que le catalogue minimal $\mathcal{C}_2$, et supposant aussi que la décomposition ne contienne aucun hexagone (dans le cas le pire), permet de borner le nombre de quadrangles par $\frac{5}{11}n$, ce qui mène à une représentation qui nécessite $\frac{91}{11}n = 8.27n$ références, avec un gain d'au moins 36%.

La taille des références

Dans une décomposition qui utilise le catalogue non minimal $\mathcal{C}'_2$, le nombre de pentagones et le nombre de quadrangles sont inférieurs à $\frac{1}{2}n$. La taille de la référence pentagone et quadrangle est alors $-1 + \log n$ bits pour référencer chaque face.

Si on impose la même taille $(-1 + \log n)$ pour les hexagones, on doit alors limiter le nombre de chaque type d'hexagone à $\frac{1}{3}n$ (on utilise deux bits dans la référence pour distinguer le type d'hexagone).

Le codage des diagonales peut toujours se faire en choisissant un ordre conventionnel pour chaque micro-triangulation (voir figure 4.3).

Un nombre de bits additionnel est nécessaire pour le codage de l'indice réciproque d'une face dans la face voisine et du type de cette face voisine, et puisque nous avons $4 + 5 + 6 = 15$ cas, ce nombre est égal à 4.

La taille des références est alors à nouveau $3 + \log n$ bits.

Le catalogue minimal $\mathcal{C}_2$ ne donne malheureusement pas de bornes qui peuvent aider à respecter la taille précédente. La taille des références faces de chaque type est $\log n$. En ajoutant les 4 bits pour les types des micro-triangulations et l'indice réciproque, la taille des références des faces est $4 + \log n$. Soit un bit supplémentaire par rapport aux autres cas vus jusqu'à présent.

4.3.3 Catalogue minimal à au moins trois triangles par micro-triangulation

La figure 4.2 illustre le catalogue minimal $\mathcal{C}_3$ qui est stable pour les opérations de mise à jour considérées. Ce catalogue contient des micro-triangulations qui ont au moins trois triangles. Le catalogue contient donc : des pentagones, des hexagones, des heptagones, des octogones, et des ennéagones. Ces micro-triangulations permettent de gagner respectivement 8, 12, 16, 20, et 24 références sur les triangles qu'ils regroupent. Ce qui produit des gains respectifs de $\frac{8}{3}$, 3, $\frac{16}{5}$, $\frac{10}{3}$ and $\frac{24}{7}$ références pour chaque triangle converti en une micro-triangulation de ce catalogue.

Le cas le pire en termes d'espace mémoire pour une décomposition en utilisant le catalogue $\mathcal{C}_3$ est obtenu quand cette décomposition ne contient que des pentagones. Le coût global de stockage dans le cas le pire est donc $\frac{23}{3}n = 7.67n$ références pour une triangulation de n points. Le gain correspondant est donc au moins égal à 41% par rapport à la représentation explicite de base.

4.4 Implémentation et résultats

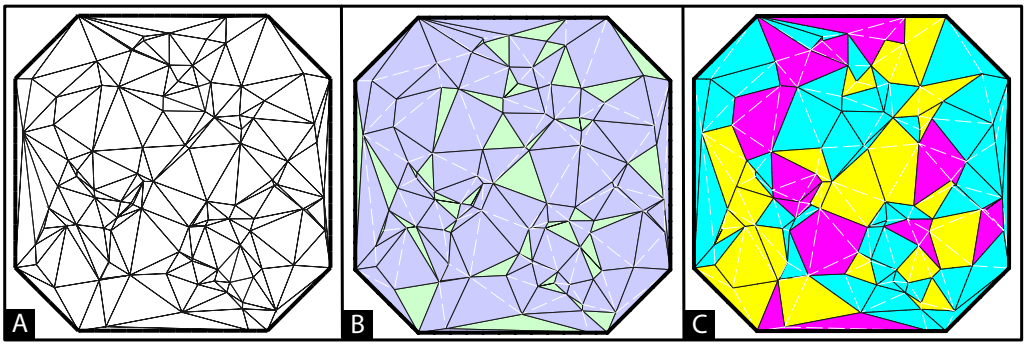

FIG. 4.10: Illustration de trois représentations d'une même triangulation (A) La triangulation à base de triangles explicite, ou la représentation par le catalogue $\mathcal{C}_1$ (B) Une représentation de la même triangulation en utilisant le catalogue $\mathcal{C}'_1$ (C) Une représentation de la même triangulation en utilisant le catalogue $\mathcal{C}_2$.

Trois catalogues sont implémentés pour pouvoir estimer l'impact pratique sur l'espace mémoire nécessaire à la représentation d'une triangulation en utilisant cette approche :

- Le catalogue trivial $\mathcal{C}_1$, qui est la représentation explicite à base de triangles.
- Le catalogue *Triangle-Quadrangle* $\mathcal{C}'_1$.
- Le catalogue minimal avec au moins deux triangles par micro-triangulation $\mathcal{C}_2$.

Le catalogue *Triangle-Quadrangle* $\mathcal{C}'_1$ est implanté comme un module de la bibliothèque *CGAL*[CGAL]. L'obligation de garder la même interface, et la même architecture logicielle a limité les optimisations en termes de temps de calcul pour ce catalogue. Ceci explique les temps de calcul relativement supérieurs par rapport au temps de calcul du troisième

catalogue. Le premier et le troisième catalogue sont implantés indépendamment de la bibliothèque *CGAL*.

Les résultats de cette implantation sont décrits dans les tableaux 4.1 et 4.2, et sont représentés sous un format graphique dans la figure 4.14.

4.4.1 Détails de l'implémentation

Les résultats dans les tableaux évaluent l'espace mémoire consommé par une triangulation de Delaunay d'un ensemble de points en dimension 2, et l'impact sur le temps de calcul en deux phases :

- **La construction de la triangulation** où on mesure le temps nécessaire pour construire la triangulation.
- **La manipulation de la triangulation** où on lance un certain nombre de requêtes de localisation dans cette triangulation (détermination de la face contenant un point donné dans cette triangulation).

La triangulation est construite incrémentalement (les points sont insérés un par un). L'insertion d'un point se fait en trois étapes :

- **localiser** le triangle qui contient géométriquement le nouveau point à insérer. Le point peut se situer dans un triangle, sur une arête, ou en dehors de l'enveloppe convexe des points déjà insérés.
- **Insérer le point** :
  - L'insertion dans un triangle se réalise en créant trois nouveaux triangles (le produit de la liaison de ce nouveau point aux sommets du triangle qui le contient), et en supprimant le triangle englobant.
  - L'insertion d'un point sur une arête est faite de la même manière, sauf que le nombre de triangles créés est quatre (produit de la liaison du point au deux sommets opposés à cette arête), et le nombre de triangles supprimés est deux (les deux triangles incidents à cette arête).
  - L'insertion en dehors de l'enveloppe convexe se fait en créant des nouveaux triangles connexes et adjacents à la triangulation.
- **Propager la propriété de Delaunay** : effectuer une série de bascules d'arêtes en partant du nouveau sommet pour rétablir la propriété du cercle vide sur toute la triangulation.

4.4.2 Conception de la structure de données

La structure de données proposée dans ce travail doit fournir l'interface définie par les spécifications suivantes :

- **Définition unique à partir d'une référence** : On doit pouvoir accéder à un

triangle en ayant sa référence ; Et de même pour un sommet de la triangulation.

- **Accès aux champs des triangles et des sommets** : Cette option garantit l'accès aux sommets et aux voisins d'un triangle donné, et à un triangle incident à un sommet donné ; et garantit aussi la possibilité de modifier ces attributs.
- **L'itération sur les triangles et sur les sommets de la triangulation** : Cette option consiste à pouvoir visiter tous les triangles ou tous les sommets de la triangulation sans aucune restriction sur l'ordre de ces objets.

Définition des références

La définition des références dans le cas du catalogue trivial $\mathcal{C}_1$ est directe. La référence d'un objet (sommet ou triangle) est le numéro de la case contenant cet objet dans le tableau associé. L'accès est alors direct, chaque référence définit un objet unique dans le tableau (voir la figure 4.12).

Dans le cas des autres catalogues, la solution est un peu plus compliquée. Une référence est du type :

$$\textit{Type} + \textit{Triangle} + \textit{Numéros}$$

où *Type* est le type de la micro-triangulation du catalogue, *Triangle* est le numéro du triangle dans cette micro-triangulation, et *Numéro* est le numéro de la case contenant l'objet dans le tableau correspondant à cette micro-triangulation (voir la figure 4.12).

L'accès à un objet de la triangulation (sommet ou triangle) à partir d'une référence nécessite la définition du type de la micro-triangulation, l'extraction de cette micro-triangulation, et la lecture du triangle voulu.

L'information du *triangle* peut être omise selon le type de l'application. Pour accéder au voisin d'un triangle par exemple, on peut se contenter de déterminer les deux sommets incidents dans la micro-triangulation voisine, et en déduire le triangle pointé.

Accès aux champs et modifications des attributs

La représentation des triangles est explicite dans le cas du catalogue trivial $\mathcal{C}_1$. L'accès aux voisins et aux sommets est par conséquent direct : l'objet en mémoire est une suite de références (trois pour les sommets et trois pour les voisins). La lecture et l'écriture se font directement sur l'objet en mémoire.

Dans le cas des autres catalogues, le triangle n'a aucune représentation explicite, on doit donc déduire les voisins d'un triangle donné à partir de la micro-triangulation qui le contient, et ce, à chaque accès à ce triangle en mode lecture. En mode écriture, la micro-triangulation correspondante est continuellement mis à jour (voir la figure 4.12).

L'accès à une face incidente à un sommet se fait de la même manière dans les deux cas vu que la représentation des sommets est similaire dans les deux cas.

Itération sur les objets

Cette option peut se réaliser par un itérateur standard sur les éléments du tableau[Muss 95].

Les tableaux utilisés dans l'implémentation sont à deux étages (voir figure 4.11) où les blocs de données sont tous de taille $\sqrt{N}$ (N est le nombre d'éléments dans le tableau). Les blocs de données sont alloués au besoin, pour minimiser l'espace alloué et non utilisé. L'itération sur un tel tableau est réalisée par des itérations successives sur tous les blocs alloués.

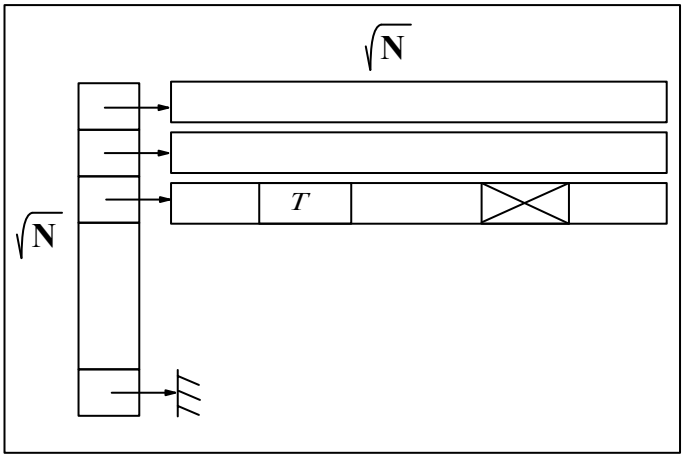

FIG. 4.11: Tableau à deux étages, où les éléments sont alloués par blocs de $\sqrt{N}$.

L'itération sur les triangles dans le cas des catalogues, se fait en choisissant un ordre entre les différentes micro-triangulations (en taille par exemple), et d'itérer successivement sur les différents tableaux associés aux micro-triangulations en énumérant à chaque lecture d'une micro-triangulation ses triangles internes.

4.4.3 Résultats pratiques

Les résultats présentés dans les tableaux 4.1 et 4.2 expriment ce qui suit :

- La taille de la mémoire résidente que la triangulation d'un nombre n de points occupe.
- Le temps nécessaire à la construction de cette triangulation, sachant que les points sont triés suivant une courbe de Hilbert[Dela 07] pour améliorer la performance de la localisation des points, ce qui permet d'accélérer la construction de la triangulation.
- Le pourcentage des différentes micro-triangulations du catalogue dans la triangulation.

Les temps de calculs pour chaque catalogue sont en deux modes :

- **mode brut**, où les points sont insérés, et la décomposition de la triangulation est mise à jour suivant un schéma identique aux cas de figures exprimés dans les figures 4.6 et 4.8 sans aucune opération supplémentaire.

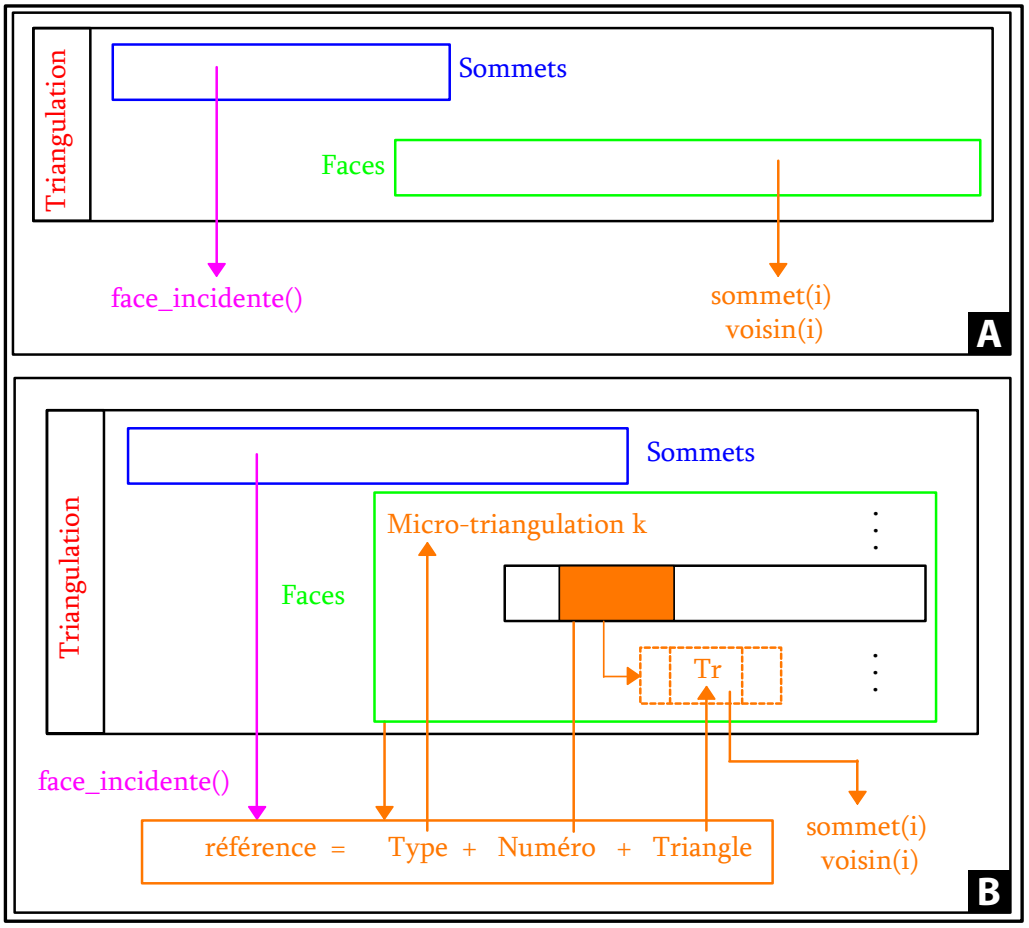

FIG. 4.12: La conception de la structure de données pour une représentation à base de catalogues. (A) l'accès aux sommets et faces d'un triangle, et à la face incidente d'un sommet est direct dans le cas du catalogue trivial (B) L'accès aux sommets et voisins d'un triangle dans le cas des autres catalogues doit passer par deux étapes : (1) Déterminer le type de la micro-triangulation englobante (2) lire cette micro-triangulation dans le tableau associé (3) Extraire le bon triangle interne (4) accéder aux attributs.

– **mode optimisé**, où on applique une opération supplémentaire d'optimisation : après chaque insertion, on parcourt le voisinage direct du nouveau point pour maximiser le nombre de micro-triangulations ayant un nombre élevé de triangles. Pour le catalogue $\mathcal{C}_1'$: les triangles voisins isolés sont regroupés deux à deux ; pour le catalogue $\mathcal{C}_2$: les suites de quadrangles sont examinées pour essayer de construire des nouveaux hexagones ou des nouveaux pentagones, et les suites de pentagones sont examinées pour essayer de construire des nouveaux hexagones (voir la figure 4.13).

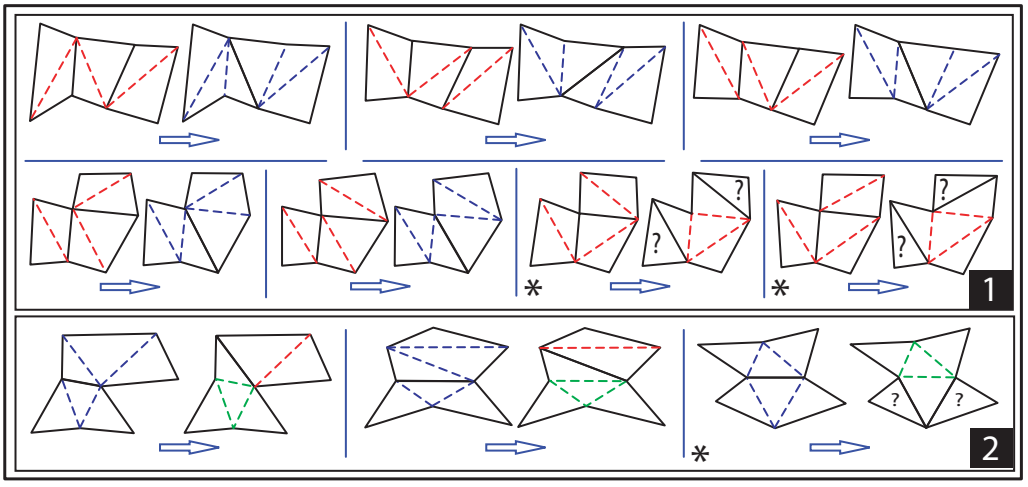

FIG. 4.13: La minimisation du nombre de micro-triangulations dans une décomposition permet de gagner plus d'espace en augmentant le nombre de micro-triangulations ayant un nombre élevé de triangles : dans le cas du catalogue $\mathcal{C}_2$ (1) les suites de triangles sont converties en pentagones ou en hexagones (2) Les suites de pentagones peuvent aussi donner naissance à des hexagones. Dans le cas où des triangles isolés peuvent apparaître (*) la conversion ne peut avoir lieu que si ces triangles peuvent être correctement collés aux micro-triangulations voisines.

4.4.4 Discussion

Les gains pratiques en termes de mémoire reflètent les estimations de calcul présentées ci-dessus. Ces gains permettent la manipulation sans utilisation de mémoires auxiliaires de jeux de données beaucoup plus grands que ceux dans le cas de la représentation à base de triangles explicite. Ceci permet également de retarder davantage le recours au transfert de données en mémoire externe (disque dur en l'occurrence).

Cependant, ce gain ne peut être obtenu sans une perte en matière de performance. Ce surcoût en temps de calcul est dû à l'indirection d'accès aux éléments introduite dans la structure de données dans le cas des catalogues pour réaliser l'interface entre les objets réels en mémoire (les micro-triangulations) et les faces (qui sont des triangles) manipulées au niveau le plus haut.

10 Millions de points (20 Millions de triangles)					
Catalogues	C_1	C'_1	C'_1 avec optimisation	C_2	C_2 avec optimisation
Triangles	19999954	8047832	2946606	-	-
Triangles	100%	40%	15%	-	-
Quadrangles	-	5976083	8526696	6046211	2296674
Quadrangles	-	60%	85%	61%	23%
Pentagones	-	-	-	2159780	2285542
Pentagones	-	-	-	32%	34%
Hexagones	-	-	-	357048	2137495
Hexagones	-	-	-	7%	43%
Gain minimum sur la combinatoire(%)	-	-	19%	-	35%
Le gain effectif sur combinatoire(%)	-	19%	26%	35%	41%
Mémoire Résidante	1,11 Go	481,1 Mo	442,1 Mo	401,6 Mo	374,3 Mo
Taux de mémoire nécessaire (%)	-	42%	39%	35%	33%
Temps	2,12 minutes	8 minutes	12,8 minutes	5,33 minutes	13,32 minutes

TAB. 4.1: Résultats sur une machine 32 bits avec un processeur Intel(R) Pentium(R) 4 Core Duo CPU 3.60 GHz avec 2Go de mémoire vive

100 Millions de points (200 Millions de triangles)					
Catalogues	$\mathcal{C}_1$	$\mathcal{C}'_1$	$\mathcal{C}'_1$ avec optimisation	$\mathcal{C}_2$	$\mathcal{C}_2$ avec optimisation
Triangles	199999950	80618230	29520044	-	-
Triangles	100%	40%	15%	-	-
Quadrangles	-	59690884	85239977	60389602	22964421
Quadrangles	-	60%	85%	60%	23%
Pentagones	-	-	-	21616898	22849256
Pentagones	-	-	-	33%	34%
Hexagones	-	-	-	3592513	21380835
Hexagones	-	-	-	7%	43%
Gain minimum sur la combinatoire(%)	-	-	19%	-	35%
Le gain effectif sur combinatoire(%)	-	19%	26%	35%	41%
Mémoire Résidante	11,2 Go	4,7 Go	4,3 Go	3,9 Go	3,6 Go
Taux de mémoire nécessaire (%)	-	42%	38%	35%	32%
Temps	12 min	50 min	1 heure 24 min	29 min	1 heure

TAB. 4.2: Résultats sur une machine 64 bits Bi-Processor Intel(R) Xeon(R) Quad Core CPU 2.33 GHz avec 16 Go de mémoire vive

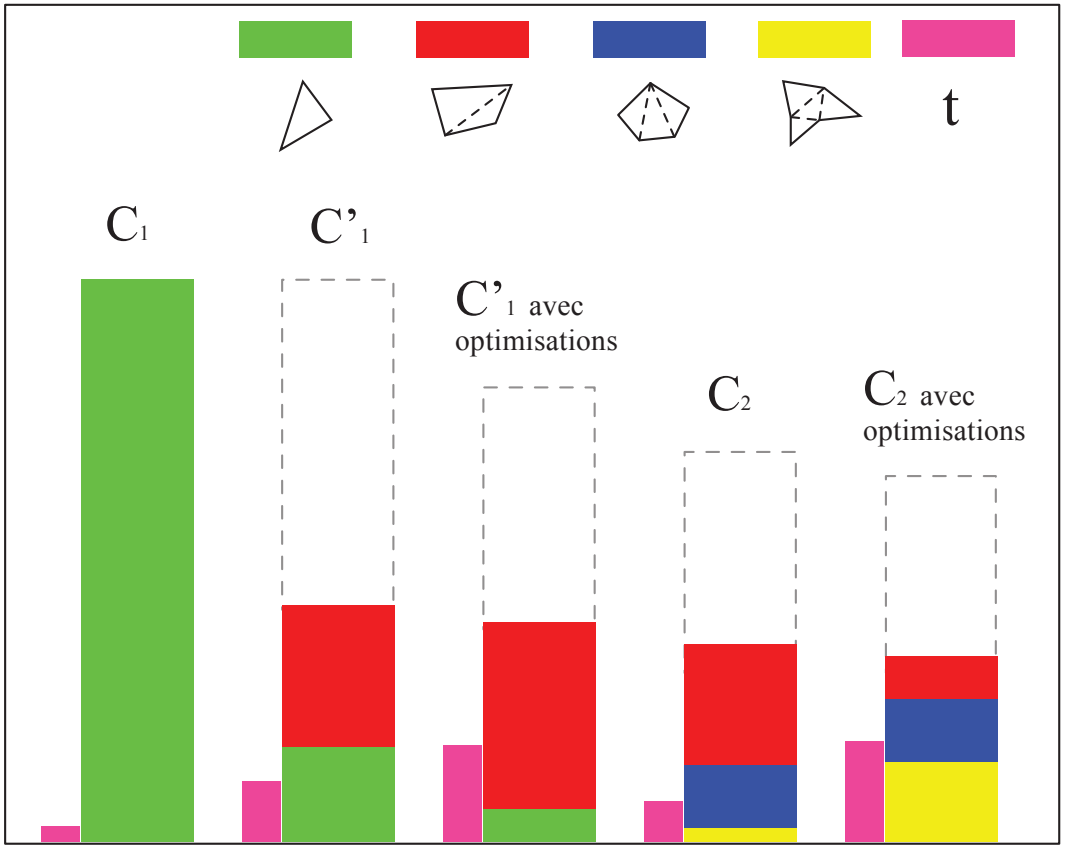

FIG. 4.14: Représentation graphique des gains apportés par l'utilisation des catalogues pour représenter les triangulations. Les résultats présentés dans les tableaux 4.1 et 4.2 sont exprimés sur cette illustration.
Les hauteurs des rectangles représentent les espaces en mémoire alloués par les différentes structures.
Les pointillés représentent les estimations de l'espace mémoire calculées dans les sections précédentes (correspondant aux gains théoriques minimaux). Les rectangles roses représentent le temps de construction de chaque structure.

Ce surcoût peut être modulé par le niveau de tassement voulu. Ceci est traduit par l'optimisation qui vise à minimiser les micro-triangulations ayant un nombre inférieur de triangles, et aussi par le nombre de triangles par micro-triangulation dans le catalogue (les micro-triangulations ayant un nombre élevé de triangles nécessitent plus de traitement pour accéder aux triangles internes).

On constate que le catalogue $\mathcal{C}_2$ réalise le meilleur taux de mémoire par rapport à la perte en temps d'exécution. Ce catalogue réduit l'espace mémoire utilisé par un facteur de 3, pour une augmentation du temps de calcul par un facteur de $2,5$.

4.5 Conclusion

Ce chapitre a été consacré à l'utilisation des catalogues stables pour représenter les triangulations en 2D. La mise à jour de la triangulation et sa construction dynamique se font en temps constant tout en maintenant une décomposition économique en micro-triangulations. Plusieurs catalogues ont été définis et étudiés en termes de calcul de bornes inférieures, et en matière de complexité pratique, en l'occurrence l'impact sur le temps de calcul.

En pratique, le gain en mémoire peut être maximisé en maintenant un ratio faible entre le nombre de micro-triangulations ayant un nombre élevé de triangles et le nombre des micro-triangulations ayant moins de triangles. Ceci se réalise en effectuant une étape d'optimisation locale après chaque opération de mise à jour. Cependant, cette phase introduit un surcoût en temps de calcul.

Chapitre 5

Codage en sous-triangulations

5.1 Introduction

L'utilisation des indices pour référencer les sommets et les faces de la triangulation en dimension 2, sur une taille contrôlée au bit près s'est avérée très coûteuse en matière de temps de calcul (voir chapitre 3). Ceci est dû en grande partie aux opérations élémentaires de lecture et d'écriture appelées en permanence, et incluant des opérations de décalage de bits qui se révèlent plutôt lentes.

Ce chapitre tente de réduire la complexité de cette technique, en alignant les références sur un nombre entier d'octets, sans être obligé à utiliser la taille du mot mémoire en entier. Ceci ne peut se faire dans le cas des grandes triangulations, sans réduire les tailles des références.

Pour ce faire, la triangulation est subdivisée en *sous-triangulations*, ce qui permet d'utiliser des références locales de taille réduite au sein de la même sous-triangulation. La taille de ces références locales dépendra bien évidemment de la taille de la sous-triangulation.

Les références de la sous-triangulation ne sont pas toutes internes, celles qui référencent des voisins n'appartenant pas à la même sous-triangulation sont obligatoirement globales, et sont de taille supérieure à la taille des références internes. Le nombre de ces références globales présente un surcoût, et est proportionnel à la taille des frontières entre les sous-triangulations.

Le gain apporté par cette méthode est relatif au taux de remplissage des sous-triangulations. Ce taux doit être maintenu élevé, pour pouvoir profiter du gain apporté par l'utilisation des références locales. Les stratégies de décomposition d'une sous-triangulation en deux ou plusieurs morceaux quand cette sous-triangulation atteint la taille maximale, ainsi que la distribution des points jouent des rôles importants en cette direction. Bien que le deuxième facteur ne puisse être contrôlé, le premier peut être ajusté pour donner des résultats assez satisfaisants.

La taille maximale des sous-triangulations est aussi un paramètre déterminant. Des

grandes sous-triangulations induisent des frontières *inter-sous-triangulations* plus petites, et donc un surcoût réduit ; Cependant, la taille des références locales est plus grande. Inversement, des sous-triangulations plus petites augmentent le coût des références globales, mais réduisent la taille des références locales.

5.2 La conception de la triangulation en plusieurs niveaux

La décomposition de triangulations est utilisée pour les représentations des triangulations[Cast 06a], ou pour la compression[Lavo 05]. L'idée consiste à partitionner les simplexes de la triangulation en plusieurs parties connexes, induisant des ensembles plus petits, ce qui permet d'utiliser moins de bits pour référencer ses éléments.

5.2.1 La structure de donnée

Une triangulation $\mathcal{T}$ est maintenue comme un ensemble de sous-triangulations $\{t_i\}$ ($i = 1..m$, où m est le nombre de sous-triangulations).

Les simplexes de la triangulation se différencient par leurs types, suivant leurs positions par rapport aux frontières des sous-triangulations. Les arêtes de la triangulation sont soit :

- des ***arêtes internes*** aux sous-triangulations.
- des ***arêtes partagées*** par deux sous-triangulations.
- des ***arêtes du bord*** de la triangulation globale.

Les sommets sont soit :

- des ***sommets internes*** aux sous-triangulations.
- des ***sommets partagés*** par deux sous-triangulations.
- des ***sommets du bord*** de la triangulation tout entière.

Enfin, les faces sont soit :

- des ***faces internes*** n'ayant ni arêtes de bord, ni arêtes partagées.
- des ***faces limites*** ayant une ou des arêtes partagées.
- des ***faces du bord*** ayant une ou des arêtes du bord.

Il est évident qu'un sommet peut être à la fois un sommet partagé, et un sommet du bord, et une face de la triangulation peut être à la fois une face limite et une face du bord.

5.2.2 Les sous-triangulations

Une ***sous-triangulation*** d'une triangulation globale $\mathcal{T}$ composée d'un ensemble de triangles T, est définie par un sous-ensemble connexe T_i de l'ensemble des triangles T, et par les sommets et les arêtes de T_i.

Toute sous-triangulation est représentée à base de triangles, comme suit :

- chaque face a trois références faces (ref_{face_locale}), et trois références sommets (ref_{sommet}).

– chaque sommet à une référence face (ref_{face_locale}).

Les relations de voisinage à l'intérieur d'une sous-triangulation sont directes et représentées explicitement, en utilisant des références locales.

Définitions

- On appelle le nombre de sommets dans une sous-triangulation, ***la taille*** de la sous-triangulation.
- On appelle le nombre de sommets maximal que la sous-triangulation peut contenir, ***la capacité*** de la sous-triangulation. Cette *capacité* correspond bien à l'espace alloué en mémoire pour la sous-triangulation en question.
- On appelle ***taux de remplissage*** τ, pour une sous-triangulation t, de capacité non nulle c, contenant $taille(t)$ éléments, le ratio $\frac{taille(t)}{c}$.

La taille des références locales ref_{face_locale} dépend de la *capacité* de la sous-triangulation. Pour augmenter le gain, tout en minimisant les surcoûts du temps de calcul, les règles suivantes sont considérées :

- **Une capacité fixe et uniforme :** La capacité est fixe pour toutes les sous-triangulations. Et l'espace mémoire associé à chaque sous-triangulation est contigu, et alloué statiquement à la création de la sous-triangulation en question. Une perte est alors inévitable lorsque la triangulation est construite incrémentalement, due au non remplissage très probable des sous-triangulations. Le choix de l'utilisation de sous-triangulations de capacités variables, où l'allocation de l'espace mémoire pour les éléments de ces sous-triangulations est dynamique, ne fait qu'augmenter la complexité de la structure, et impose l'ajout d'informations supplémentaires pour spécifier la capacité de la sous-triangulation, et marquer la disposition des éléments en mémoire.
- **Des références locales sur *n octets* :** La capacité est choisie de manière à ce que le nombre de bits nécessaire à la représentation des références locales soit un multiple de 8 bits, pour éviter un gaspillage inutile de bits, ou le recours à des représentations au bit près augmentant ainsi la complexité des méthodes d'accès.

5.2.3 La connectivité inter-sous-triangulations

Lorsque la triangulation est composée de plusieurs sous-triangulations, les références locales ne suffisent plus pour représenter la topologie globale. Les bords inter-sous-triangulations nécessitent un autre type de références permettant le passage d'un triangle à un voisin n'appartenant pas à la même sous-triangulation.

Le graphe des sous-triangulations n'est pas représenté explicitement, ni d'ailleurs le graphe intra-sous-triangulation (qui est le graphe interne à une sous-triangulation). C'est

un schéma combinatoire unique qui représente toute la triangulation, pour pouvoir bénéficier du gain que peut apporter l'utilisation des références locales.

Les faces de la triangulation gardent la même représentation, et par conséquent le même type de références, qu'elles soient des faces internes, des faces limites, ou des faces du bord.

On considère une triangulation de n sommets, représentée par une décomposition en sous-triangulations de m sommets chacune. Le nombre de triangles dans chaque sous-triangulation est borné par $2m$. Une face est alors représentée par :

- 3 références sommets de taille $\log m$ bits.
- 3 références faces de taille $1 + 1 + \log m$ bits.

La taille des références des sommets est égale à $\log m$ bits, car tous les sommets sont référencés par des faces de la même sous-triangulation, la référence est donc locale dans tous les cas.

La taille des références des faces dans les sommets est égale à $1 + \log m$ bits, car on peut toujours imposer que les faces référencées par les sommets soient de la même sous-triangulation.

La taille des références de faces voisines est égale à $1+1+\log m$ bits. Le bit additionnel détermine l'interprétation de la référence en séparant deux cas :

- **La référence est locale :** dans ce cas-là, cette référence renvoie un élément de la même sous-triangulation, qui est déréférencé par le numéro représenté par les $1 + \log m$ bits restants.
- **La référence est globale :** la référence renvoie un élément dans une autre sous-triangulation. Dans ce cas-là, les $1 + \log m$ bits restants renvoient l'indice d'une case dans un tableau spécial ; et cette case contient la référence globale, qui est une composition de deux parties, le numéro de la sous-triangulation en question t_j, et le numéro de l'élément déréférencé dans cette sous-triangulation.

Par conséquent, la taille des références des sommets et des faces est bornée par $2 + \log m$.

5.2.4 Le tableau des références globales

Pour gérer les références globales, un tableau supplémentaire doit être associé à chaque sous-triangulation. Les éléments de ce tableau sont des références globales de la taille de $\log\left(\frac{n}{m}\right) + 1 + \log(m) = 1 + \log n$. La taille de ce tableau dépend de la taille du bord de la sous-triangulation, ce qui rend la gestion de ce tableau complexe et imprévisible.

Une alternative consiste à exploiter la formule suivante, déduite de la formule d'Euler-Poincaré[Bond 76, Dies 00] :

$$2m - 2 = t + k; \tag{5.1}$$

Où t est le nombre de triangles, k est le nombre des triangles du bord. Il est clair que la

somme du nombre de faces de la sous-triangulation, et du nombre des arêtes du bord (qui est égal au nombre des références externes ou globales de la sous-triangulation) est bornée par $2m$.

Supposons que la taille des références globales ($1 + \log n$ bits) est inférieure ou égale à la taille d'une face de la triangulation ($6 + 6 \log m$ bits), ce que l'on garantit en pratique par le choix de m ; On peut alors utiliser un même tableau de taille $2m$ (exprimée en nombre de faces) pour garder les faces de la sous-triangulation et les références du bord (voir figure 5.1).

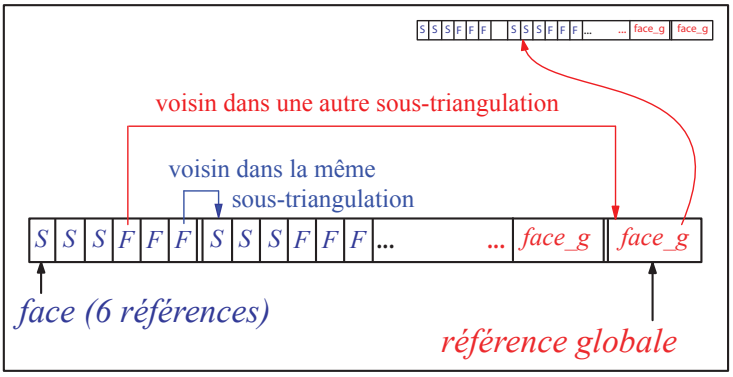

FIG. 5.1: Un seul tableau de taille $2m$ triangles suffit pour représenter à la fois les faces de la sous-triangulation, en partant de la gauche ; et les références externes de cette même sous-triangulation vers les autres sous-triangulations en partant de la droite.

5.2.5 Le coût de la structure

Idéalement, la triangulation est subdivisée en $\frac{n}{m}$ sous-triangulations, de m points chacune. Chaque sous-triangulation utilise deux tableaux :

- Un tableau de m sommets, chacun représenté par une référence face de $2+\log m$ bits.
- Un tableau de $2m$ faces, chacune représentée par $6 + 6 \log m$ bits.

Le coût global est de $14m+13m \log m$ bits. Le coût équivalent pour une représentation explicite serait $7m+13m \log n$. Le facteur entre les deux modes de représentation est donc essentiellement de l'ordre de $\frac{logm}{\log n}$. Ceci peut nous amener à minimiser au maximum la capacité des sous-triangulations pour pouvoir obtenir le meilleur facteur de gain, ce qui revient à choisir des sous-triangulations à triangle unique.

Cependant, il y a un autre facteur très influent dans le calcul du coût, qui est la taille du bord des sous-triangulations. Il faut noter tout d'abord, que la décomposition de la triangulation n'est pas une partition. Ceci revient à dire qu'il y a des sommets partagés entre les sous-triangulations. Et choisir des sous-triangulations à triangle unique, revient à partager tous les sommets, et par conséquent, rendre toutes les références externes, ce qui prive la méthode de son intérêt.

On ne peut pas non plus maximiser le nombre de sommets internes, ce qui produit la coïncidence de la référence locale avec la référence globale, et de même, nous conduit à la représentation explicite globale.

C'est donc un compromis à gérer entre la capacité des sous-triangulations, et le nombre de sommets internes.

Le coût de la représentation dépend d'autres facteurs, comme la distribution des points, et les stratégies de construction et de mise à jour de la triangulation.

La disposition des points joue un rôle primordial dans la détermination de la décomposition possible d'une triangulation en sous-triangulations. Dans notre travail, on considère que les points sont généralement *bien* distribués (ce qui veux dire que les points sont uniformément distribués).

Pour ce qui est de la stratégie de construction et de mise à jour, la section suivante y est entièrement dédiée.

Cas pratique Considérons maintenant le cas d'une triangulation de n points, où les références faces et sommets sont de taille 7 bits. Ceci nous permet d'avoir des sous-triangulations de taille $2^6 = 64$ sommets chacune. Un bit est réservé dans chaque octet à l'usage interne du tableau (pour le marquage des cases libres).

Si toutes les sous-triangulations sont pleinement remplies (ignorant que les sommets se partagent), le gain est de $\frac{3}{4} = 75\%$ par rapport à la représentation classique. Ceci est déduit du fait qu'on passe d'une représentation des références sur des entiers de 4 octets à des références d'un octet.

Toutefois, les sous-triangulations ne sont pas toujours pleinement remplies. En moyenne, elles sont remplies à $\frac{3}{4} = 75\%$ (au mieux, le taux de remplissage est à 100%, et au pire, il est à 50%, lorsque une sous-triangulation atteint la limite, elle est subdivisée en deux sans que les deux moitiés ne reçoivent de nouvelles insertions), ce qui nous donne un gain moyen de $\frac{2}{3} = 66\%$.

5.3 La construction et la mise à jour de la triangulation

La triangulation est construite incrémentalement, c'est à dire que les points sont insérés un à un. Et les sous-triangulations sont, par conséquent, construites l'une après l'autre.

Tout au début, la triangulation ne contient qu'une seule sous-triangulation, où toutes les opérations sont entreprises comme s'il n'y avait qu'une seule triangulation. Dès que la taille de cette sous-triangulation atteint la capacité de la sous-triangulation, une nouvelle sous-triangulation est créée pour permettre l'insertion de nouveaux points. Et la construction de la triangulation continue jusqu'à l'insertion de tous les points.

5.3.1 Insertion d'un nouveau point

L'algorithme d'insertion est celui de la bascule d'arêtes, présenté dans la section 1.2.2. On considère que la capacité d'une sous-triangulation est c, et que le nombre d'éléments d'une sous-triangulation t à un instant donnée est $taille(t)$. Les cas de figures qui se présentent sont comme suit :

Insertion dans une face f Cette insertion produit deux faces supplémentaires dans la sous-triangulation t en question. Si l'insertion de ces deux faces ne déborde pas la capacité de la sous-triangulation, ce qui revient à $taille(t)+2 \leq c$, l'insertion se réalise normalement.

Si par contre, la sous-triangulation déborde après l'insertion des deux nouvelles faces ($taille(t) + 2 > c$), cette sous-triangulation est subdivisée d'abord en deux nouvelles sous-triangulations t_1 et t_2, et l'insertion se fait dans la nouvelle sous-triangulation contenant la face f.

Les stratégies de décomposition seront détaillées dans la section 5.4

Insertion sur une arête L'insertion sur une arête produit aussi deux nouvelles faces, le même raisonnement est suivi pour la mise à jour de la décomposition de la triangulation.

L'insertion sur une arête peut bien évidemment se faire sur la frontière entre deux sous-triangulations. Ce cas ne se pose pas en pratique, vu que l'insertion sur une arête est détournée combinatoirement en combinant l'insertion dans une face avec une bascule d'arête (voir figure 1.3).

5.3.2 La bascule d'arête

Pour une arête interne à une sous-triangulation, la bascule se réalise normalement. Cependant, pour une *arête partagée* entre deux sous-triangulations, la mise à jour de la décomposition n'est pas évidente.

La bascule d'une telle arête se fait en passant l'une des deux faces à la sous-triangulation voisine. Le choix est fait en fonction des tailles des deux sous-triangulations. La face de la sous-triangulation la plus grande est transférée à l'autre sous-triangulation.

Le cas restant est, bien évidemment, la bascule d'arête en interne dans la même sous-triangulation.

Définition On dit qu'une face f de la triangulation est *une face d'étranglement* dans sa sous-triangulation t, si la suppression de cette face, engendre une singularité (voir section 1.1.2) dans la sous-triangulation, ou divise la sous-triangulation en deux composantes connexes, ce qui d'ailleurs, fausse l'équation 5.1 (voir figure 5.2).

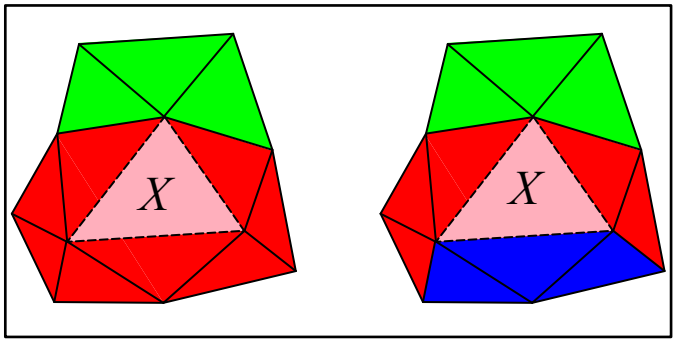

FIG. 5.2: Le triangle X est une face d'étranglement dans la sous-triangulation rouge, c'est un triangle dont la suppression génère une singularité.

La bascule d'arête entre deux sous-triangulations t_1 et t_2, nécessite un traitement spécifique lorsque ce transfert n'est praticable dans aucun des deux sens. La non possibilité de transférer *une face limite* de sa sous-triangulation (*la sous-triangulation source*) à une sous-triangulation voisine adjacente (*la sous-triangulation de destination*) peut se produire dans deux cas :

- La face en question est une *face d'étranglement*.
- La sous-triangulation voisine est pleine (sa taille est égale à sa capacité).

Dans ce cas là, deux solutions sont à appliquer :

Décomposition de la sous-triangulation de destination

Lorsque le problème se limite à la taille de la sous-triangulation de destination, ceci peut être résolu par la décomposition de cette dernière en suivant l'un des schémas proposés dans la section suivante.

Décomposition à partir d'une face d'étranglement

Dans le cas où l'arête à basculer est adjacente à une face d'étranglement f (ou à deux faces d'étranglement f_1 et f_2), et que la face opposée ne peut être transférée, une décomposition spéciale doit être entreprise. La sous-triangulation t contenant la face d'étranglement est décomposée en deux sous-triangulations t_1 et t_2, de telle sorte que la face f appartienne à la sous-triangulation t_1, et est adjacente à la sous-triangulation t_2. Après cette décomposition, la face f n'est plus une face d'étranglement dans sa sous-triangulation, et peut être transférée à l'une de ces sous-triangulations voisine (voir figure 5.3).

Les deux solutions peuvent être combinées dans certains cas pour permettre la bascule d'arête. Par exemple, lorsque les deux sous-triangulations sont pleines, et que les deux faces incidentes à l'arête à basculer sont des faces d'étranglement.

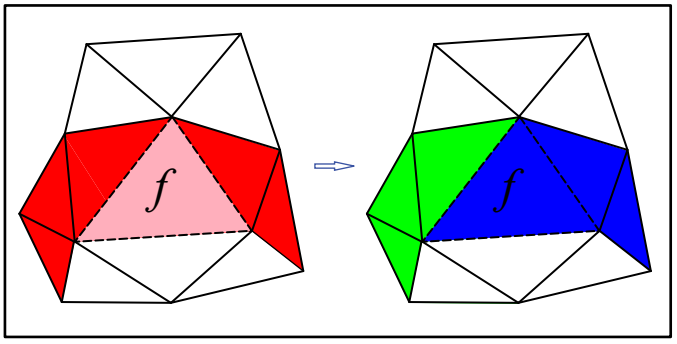

FIG. 5.3: La sous-triangulation rouge est décomposée en deux sous-triangulations (verte et bleue) en partant de la face d'étranglement f. Après la décomposition, la face f n'est plus une face d'étranglement.

5.4 Stratégies de décomposition

Lorsque la sous-triangulation atteint la taille maximale autorisée, elle doit être subdivisée en deux parties. Pour ce faire, on définit un ensemble de faces à supprimer de la sous-triangulation et à insérer dans la nouvelle sous-triangulation.

5.4.1 Une décomposition progressive

Pour décomposer une sous-triangulation t en deux sous-triangulations, on parcourt l'arbre couvrant de la sous-triangulation, et on transfère les triangles un par un à la nouvelle sous-triangulation. L'arbre couvrant n'est pas construit, le parcours est implicite et direct. Le schéma général pour une décomposition progressive est décrit dans le pseudocode 5.1.

Les pseudocodes 5.2 et 5.3 décrivent les décompositions d'une sous-triangulation t suivant cette stratégie en utilisant deux parcours différents de la triangulation (en profondeur et en largeur, en partant du sommet). La procédure crée une nouvelle sous-triangulation, et supprime les triangles de l'ancienne sous-triangulation l'un après l'autre, en les insérant au fur et à mesure dans la nouvelle.

Pour ce faire, une face du bord doit être désignée pour commencer la recherche, le choix de cette face peut être aléatoire, ce qui mène à des décompositions différentes. On peut envisager plusieurs heuristiques pour sélectionner la première face, sur le bord :

- Choisir la face extrême suivant l'axe d'allongement de la triangulation (c'est ce qui est choisi dans l'implémentation).
- Choisir la face avec le sommet ayant le degré minimal

```
Subdiviser(sous-triangulation t)
début
    // initialisation
    triangle f = t.sélectionner_une_face ;
    sous-triangulation t_bis = créer_une_nouvelle_sous-triangulation() ;
    liste<triangle> liste_t ;
    // Remplir la nouvelle sous-triangulation
    tant que(taille(t_bis) < taille(t))
        transférer(f, t, t_bis) ;
        si(liste_t.est_vide)
            arrêter_la_décomposition() ;
        fin si
        f = liste_t.suivant() ;
    fin tant que
fin.
```

Pseudo 5.1: Algorithme de décomposition progressive de sous-triangulations.

```
Subdiviser_2(sous-triangulation t)
début
    // initialisation
    triangle f = t.sélectionner_une_face_du_bord() ;
    sous-triangulation t_bis = créer_une_nouvelle_sous-triangulation() ;
    liste<triangle> liste_t ;
    // Remplir la nouvelle sous-triangulation
    tant que(taille(t_bis) < taille(t))
        transférer(f, t, t_bis) ;
        si(triangle_gauche(f) est dans t)
            liste_t.insérer_à_la_fin(triangle_gauche(f)) ;
        fin si
        si(triangle_droit(f) est dans t)
            liste_t.insérer_à_la_fin(triangle_droit(f)) ;
        fin si
        transférer(f, t, t_bis) ;
        f = liste_t.début() ;
    fin tant que
fin.
```

Pseudo 5.2: Algorithme de décomposition de sous-triangulations par parcours en largeur d'abord.

```
Subdiviser_2(sous-triangulation t)
début
    // initialisation
    triangle f = t.sélectionner_une_face_du_bord() ;
    sous-triangulation t_bis = créer_une_nouvelle_sous-triangulation() ;
    liste<triangle> liste_t ;
    // Remplir la nouvelle sous-triangulation
    tant que(taille(t_bis) < taille(t))
        transférer(f, t, t_bis) ;
        si(triangle_droit(f) est dans t)
            liste_t.insérer_au_début(triangle_droit(f)) ;
        fin si
        si(triangle_gauche(f) est dans t)
            liste_t.insérer_au_début(triangle_gauche(f)) ;
        fin si
        f = liste_t.début() ;
    fin tant que
fin.
```

Pseudo 5.3: Algorithme de décomposition de sous-triangulations par parcours en profondeur d'abord.

5.4.2 Un parcours par balayage

Dans le but d'obtenir des sous-triangulations équilibrées, un parcours par balayage pour extraire les triangles à insérer dans la nouvelle sous-triangulation peut être plus efficace. La triangulation est parcourue de gauche à droite ou de haut en bas (suivant l'axe le plus long) par une ligne, et à chaque fois que la ligne franchit un triangle, ce triangle est transféré à la nouvelle sous-triangulation. Cette technique est analogue à plusieurs algorithmes en géométrie algorithmique comme pour le calcul de la triangulation de Delaunay, ou le calcul des arrangements de segments[Chen 96].

5.4.3 L'ajustement du bord d'une sous-triangulation

La décomposition progressive à partir d'une face sur le bord ne conduit pas toujours à une décomposition équitable. En effet, la décomposition ne s'arrête pas toujours lorsque les deux sous-triangulations sont de tailles équivalentes, mais peut s'arrêter prématurément. Ce cas se produit quand le parcours se trouve sur une face d'étranglement dans la sous-triangulation, et résulte souvent une sous-triangulation allongée ayant un bord très large, ce qui induit un taux de remplissage très faible. Les résultats obtenus ainsi peuvent être améliorés en appliquant un traitement supplémentaire.

Ce traitement vise deux buts :

- Maximiser l'équilibre de taille entre les sous-triangulations.
- Minimiser au maximum la taille du bord de la sous-triangulation, c'est à dire le nombre d'arêtes partagées avec les sous-triangulations voisines.

Pour ce faire, les triangles du bord de la sous-triangulation sont parcourus. Et chaque fois qu'on trouve un triangle ayant deux arêtes partagées avec la même sous-triangulation, ce triangle est transféré vers cette sous-triangulation. Ceci permet de maximiser le nombre de ses sommets internes, et minimiser en même temps la taille du bord de la sous-triangulation courante.

Cette opération est réalisée avant chaque subdivision. Si ce traitement n'aboutit pas à une réduction de la taille de la triangulation, la subdivision aura lieu. Cette étape d'optimisation de la forme de la sous-triangulation peut être réalisée aussi après la décomposition pour les deux sous-triangulations : l'ancienne et la nouvelle.

5.5 Quelques remarques sur la conception

Le but de ce travail est de concevoir une structure de données économique en mémoire, sans trop perdre en efficacité. Par conséquent, la simplicité et la réduction au maximum des niveaux d'indirection sont primordiales.

Certains niveaux de représentation sont admis dans le cas ou des vrais pointeurs sont utilisés, du fait que le coût en temps d'exécution est négligeable. Cependant, lorsqu'il s'agit de séparer la représentation mémoire (ce qui est stocké en mémoire) de l'interface utilisateur, il est conseillé de limiter au maximum les couches inutiles. Car dans ce modèle là, des accès multiples sont faits à chaque accès aux éléments de la structure en lecture, comme en écriture (voir chapitre 3 pour plus de détails).

5.6 Mise en œuvre.

Cette section détaille la conception de la triangulation en termes de structures de données, en décrivant les différents niveaux :

- Le niveau le plus haut : l'interface avec l'utilisateur.
- Le niveau de la structure de données.
- Le niveau de stockage.

5.6.1 L'interface

La triangulation doit garder son interface classique vis à vis de l'utilisateur. Ceci implique une invariance par rapport au modèle imposé au début de notre travail, à savoir :

- Construction d'une triangulation : par le constructeur par défaut, et le constructeur par copie.
- Insertion d'un point dans la triangulation.
- Suppression d'un sommet de la triangulation.
- Bascule d'une arête de la triangulation.
- Parcours des sommets et des triangles de la triangulation.

5.6.2 Le niveau de la structure de données

Cette couche est intermédiaire entre la couche la plus basse et la couche d'interface avec l'utilisateur. Pratiquement, une classe appelée (*Container*) est créée pour regrouper toutes les fonctionnalités et les méthodes nécessaires à la création et à la manipulation des objets.
On définit dans cette classe ce qui suit :

Les itérateurs et leurs types de valeurs

La partie cruciale de cette phase, est la définition des types *Vertex*, *Face*, *Vertex_iterator*, et *Face_iterator* ; associés respectivement aux sommets, faces, itérateurs sur les sommets, et itérateurs sur les faces.

Les faces (triangles) stockées en mémoire ne contiennent que des informations locales, et pour pouvoir les manipuler comme des triangles réels, elles ont besoin d'inclure :

- le numéro de la sous-triangulation.
- les références globales pour les champs pointant vers des faces externes.

Les faces sont donc représentées comme suit :

- La structure contient un pointeur vers la face dans le tableau, le numéro de la sous-triangulation en question, et une référence du tableau global (en cas d'utilisation de plusieurs triangulations).
- Adapter toutes les méthodes d'accès aux sommets et aux voisins à cette définition.
- Le seul inconvénient est que les faces implémentées ainsi ne sont plus utilisables en soi. On ne peut manipuler que les faces réelles (qui sont déjà dans la triangulation, i.e. insérées dans le tableau). On ne peut donc plus déclarer une face sans qu'elle ait une image dans le tableau. Ce qui paraît un peu restrictif au niveau de l'implémentation, mais en pratique, c'est tout à fait cohérent. Car on peut envisager de définir un type *Triangle*, pour en faire usage dans une telle situation, et garder le type *Face* pour les faces de la triangulation (ayant un voisinage).

Cette définition est le type de valeur de l'itérateur sur les faces. Cet itérateur est quasiment identique à son type de valeur, en le surchargeant par les opérateurs d'incrémentation et de decrémentation. On peut donc itérer sur tous les éléments, via ces itérateurs, en tenant en compte le passage d'une petite triangulation à une autre.

Les définitions des sommets et des itérateurs sur ces sommets sont analogues à ceux présentés pour les faces.

5.6.3 Le niveau de stockage

Le niveau le plus bas est celui qui représente la mémoire ou l'espace de stockage.
La taille des pointeurs utilisés pour représenter toutes les références de la triangulation (sommets et faces) est *un octet*. Ce qui implique de construire des structures de données reflétant cette réalité.
On distingue à ce stade les types et les instances suivants :

- Les *sommets de base* : les sommets où les références de faces incidentes sont codées sur un octet, et contiennent évidemment l'information géométrique (les points).
- Les *faces de base* : les structures où les champs (sommets et faces) sont codés sur un octet chacun. Ces faces ne peuvent avoir une existence en soi, parce que les champs de voisins ne déréférencent pas toujours des triangles, mais peuvent être des indices dans le même tableau dont la case correspondante contient la vraie référence du voisin en question.
- Les *tableaux de sommets et de faces* : la triangulation est subdivisée en sous-triangulations.

Chacune contient un tableau de sommets et un tableau de faces. Les sommets contiennent toujours des références locales (les références de faces incidentes). Par contre, les faces peuvent contenir des références vers des faces externes (ne faisant pas partie de la sous-triangulation courante). Pour pallier ce problème, la face en question garde un indice sur un octet dans le champ qui correspond à la face externe. Cet indice est un numéro de case dans la sous-triangulation courante. Et cette case contient la référence réelle de la face externe (une référence sur la taille du mot mémoire de la machine, contenant le numéro de la sous-triangulation, et le numéro interne de la face dans cette sous-triangulation).

– Les *itérateurs de base* : les itérateurs locaux aux petits tableaux de sommets et de faces.

5.7 Résultats et implémentation

5.7.1 Expérimentations

Les résultats dans le tableau 5.1 sont obtenus en calculant une triangulation de Delaunay d'un million de points, générés aléatoirement suivant une loi uniforme. La triangulation est calculée en utilisant l'algorithme de la bascule d'arête décrits dans la section 1.2.2. Les résultats présentent plusieurs paramètres :

– Le nombre de sous-triangulations dans la triangulation.
– Le facteur de redondance des sommets. Ce facteur est le ratio entre le nombre de points de la triangulation, et la somme des sommets des sous-triangulations. Et donne une estimation du nombre de sommets partagés entre les sous-triangulations.
– Le nombre de subdivisions (1) : ce nombre est égal au nombre de fois ou une sous-triangulation est subdivisée parce qu'elle a atteint la taille maximale.
– Le nombre de bascules d'arêtes établis pendant la construction de la triangulation.
– Le nombre de subdivisions (2) : ce nombre est égal au nombre de fois ou une sous-triangulation est subdivisée à partir d'une face d'étranglement pour permettre la bascule d'arête.
– Le nombre de subdivisions (3) : ce nombre est égal au nombre de fois ou une sous-triangulation est subdivisée parce qu'elle a atteint la taille maximale pour permettre la bascule d'arête.
– Le taux de remplissage (1) : la moyenne des taux de remplissage des tableaux de sommets.
– Le taux de remplissage (2) : la moyenne des taux de remplissage des tableaux de triangles.
– Le gain en termes de tailles de structures, qui est le facteur entre les deux tailles de structures précédentes.

Les configurations évaluées dans le tableaux correspondent aux différentes stratégies de parcours lors de la décomposition d'une sous-triangulation, et sont ci-dessous :

- Parcours par balayage :
  - Heuristique 1 : parcours par balayage simple.
  - Heuristique 2 : ajustement du bord avant la subdivision de la sous-triangulation dans le but d'équilibrer les tailles.
  - Heuristique 3 : ajustement du bord avant et après la subdivision de la sous-triangulation.
- Heuristique 4 : Parcours en largeur d'abord.
- Heuristique 5 : Parcours en profondeur d'abord.

Ces résultats sont présentés sous un format graphique dans la figure 5.4.

	Heur. 1	Heur. 2	Heur. 3	Heur. 4	Heur. 5
Nombre de sous-triangulations	54675	42299	41533	43738	47592
Facteur de redondance des sommets	1,683	1,577	1,556	1,553	1,615
Nombre de subdivisions (1)	45537	33924	34190	36569	37482
Nombre de bascules	3075543	3075543	3075543	3075543	3075543
Nombre de subdivisions (2)	9121	8369	7334	7166	12454
Nombre de subdivisions (3)	10211	10698	9454	9453	10102
Taux de remplissage (1)	51,296 %	62,131 %	62,453 %	59,191 %	56,558 %
Taux de remplissage (2)	30,483 %	39.402 %	40,129 %	38,106 %	35,020 %
gain de la structure	17,988 %	36,552 %	37.705 %	34,393 %	28,612 %

TAB. 5.1: Statistiques d'utilisation de la mémoire 5.7.1. Ces résultats sont obtenus sur une machine 32 bits Processor Genuine Intel(R) CPU T2300 1.66 GHz Core Duo avec 2 Go de mémoire vive. Les coordonnées sont en format *flottant* et représentées sur 4 octets chacune. Le gain dans le tableau est exprimé par rapport au coût de représentation d'une représentation à base de triangles explicite.

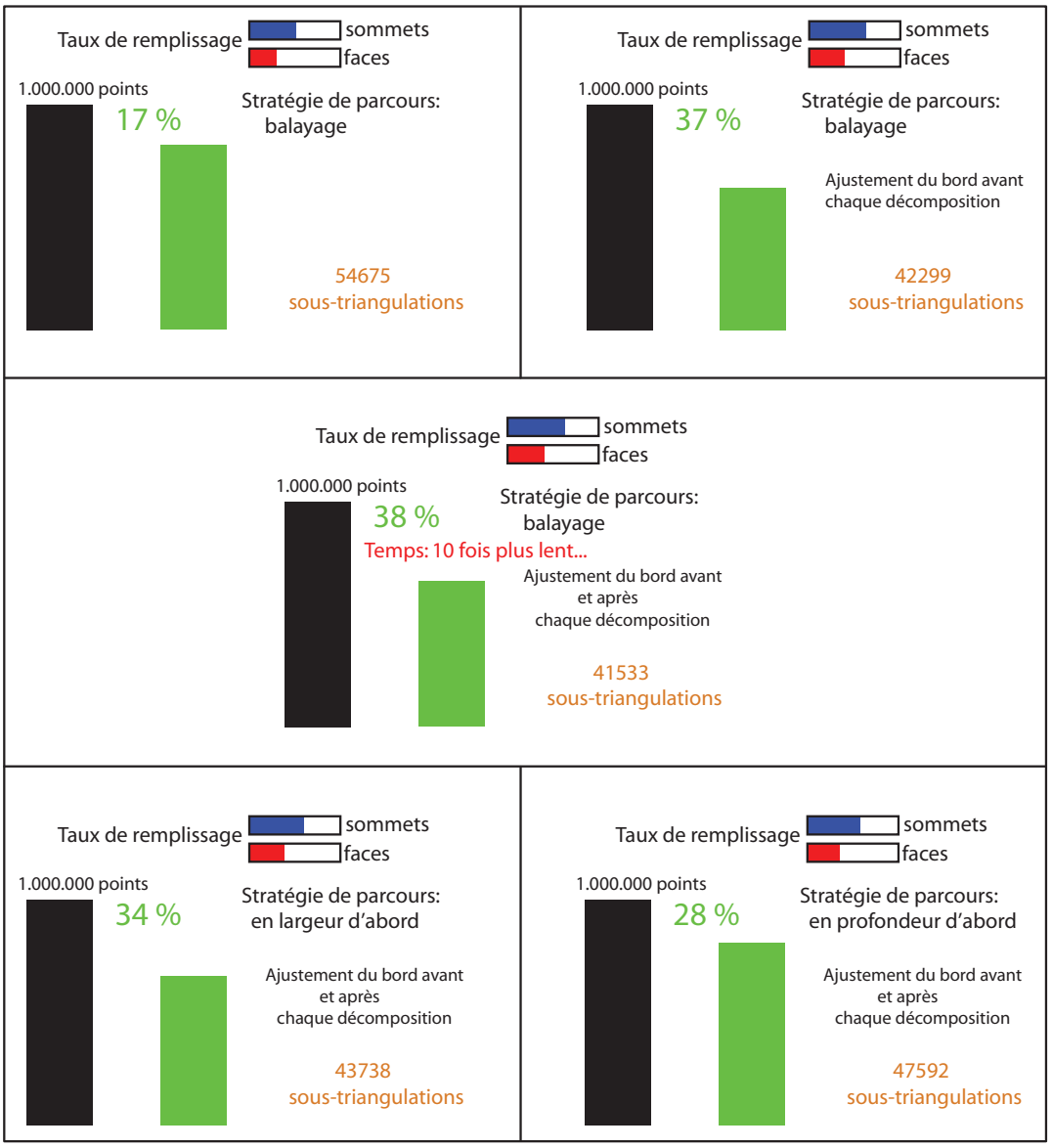

FIG. 5.4: Représentation graphique des gains apportés par l'utilisation des sous-triangulations pour représenter les triangulations. Les rectangles noirs représentent l'espace mémoire alloué par la structure de base. Les rectangles verts représente l'espace mémoire alloué par la structure à sous-triangulations.

5.8 Conclusion

Ce chapitre a été consacré à la représentation des triangulations en utilisant des sous-triangulations, pour pouvoir bénéficier de l'utilisation des références locales sur un nombre réduit de bits. La stratégie de mise à jour de la triangulation après l'application de chaque opération de mise à jour détermine les taux de remplissage des différents tableaux. Ceci détermine le gain apporté par la méthode. Ces stratégies doivent maximiser le nombre de sommets internes tout en réduisant les tailles des frontières entre les différentes sous-triangulations. Cette technique peut être combinée avec d'autres techniques tel que le codage par différences. En effet, l'allocation statique des tableaux des sous-triangulations fournit une plage d'étiquettes limitée. Les objets peuvent alors être insérés tout en imposant un étiquetage cohérent minimisant les différences entre les sommets voisins.

Conclusion

Conclusion générale

Cette thèse a été entamée dans le but de concevoir des solutions pratiques et exploitables en réponse aux problèmes de limitations d'espace mémoire lors de la manipulation de grandes triangulations. Les travaux entrepris ont suivi trois axes :

1. L'utilisation des indices comme références, pour économiser un nombre de bits sur chaque référence utilisée par la structure.

2. La définition des catalogues stables pour représenter les triangulations, dans le but de réduire le nombre des références multiples vers les mêmes sommets, et les références réciproques entre voisins.

3. La décomposition de la triangulation pour pouvoir utiliser des références locales de tailles réduites par rapport à la taille des références absolues.

Les indices pour remplacer les pointeurs

La première idée pour réduire la taille de l'espace mémoire, est de limiter aux maximum le gaspillage des bits, en représentant les références des objets de la triangulation par des indices occupant exactement le nombre de bits nécessaire. Cette idée, bien que triviale sur le plan théorique, nécessite une attention particulière lors de sa mise en œuvre.

Le coût d'une telle structure s'est avéré très élevé en termes de temps d'exécution. Ceci est dû à l'accès répétitif aux cases mémoires contenant les informations relatives aux sommets et aux objets géométriques. Cet accès est réalisé à chaque écriture, et à chaque lecture pour garder l'intégrité et la cohérence entre les variables manipulées par l'utilisateur et les données réelles en mémoire.

L'implantation à été conçue suivant le schéma de la bibliothèque *CGAL*, en gardant toutes les fonctionnalités et l'interface proposée par les triangulations de *CGAL*. Cette contrainte a limité la marge pour les optimisations en termes de temps de calcul. Cependant, le gain en mémoire n'est pas négligeable, surtout pour les machines à 64 *bits*.

Les catalogues stables

L'idée des catalogues stables est de regrouper les triangles en micro-triangulations. Ces micro-triangulations sont des structures redondantes, et la triangulation est alors conçue comme une subdivision de ces structures. Ces micro-triangulations sont définies dans des catalogues qui doivent être stables pour les opérations utilisées par la structure de données afin de garantir la construction et la mise à jour de la triangulation.

L'utilisation des catalogues stables permet d'éliminer quelques références vers les sommets et entre les voisins. Le nombre minimal de triangles par micro-triangulation détermine le gain en mémoire. Les catalogues ayant un nombre élevé de triangles par micro-triangulation réalisent des gains très intéressants.

Cependant, cette décomposition induit un niveau d'indirection supplémentaire, lorsque les triangles sont consultés, et/ou modifiés. Les résultats sont prometteurs, les gain pratiques sont conformes aux estimations de calculs établis suivant certaines contraintes. Ces gains sont obtenus tout ayant une augmentation limitée du temps de calcul.

La décomposition en sous-triangulations

Dans cette partie, les références ne sont plus des pointeurs absolus sur la taille du mot mémoire de la machine, mais des indices sur un nombre de bits multiples de 8. Ce qui nous épargne quelques bits pour chaque référence. D'un point de vue architectural, la triangulation est décomposée en sous-triangulations de tailles moyennes.

Les références entre voisins ne sont donc pas de même nature. Les références entre différentes sous-triangulations nécessitent un traitement spécifique. Ces références globales sont stockées dans la même structure utilisée pour stocker les faces de la sous-triangulation.

Plusieurs facteurs ont été étudiés pour améliorer les taux de remplissages des sous-triangulations, qui sont liés au gain apporté par la méthode. Le passage le plus important est la stratégie de décomposition d'une sous-triangulation en plusieurs sous-triangulations. Le gain est amélioré aussi par des heuristiques visant à maximiser le nombre de sommets internes aux sous-triangulations. Ces heuristiques sont appliquées sur les bords des sous-triangulations en appliquant des transferts de triangles entre les sous-triangulations voisines.

Travaux futurs

Le sujet n'est toujours pas traité en entier, surtout que les tailles de triangulations ne cessent d'augmenter. Des solutions pratiques ont été proposées dans cette thèse, et méritent un suivi avec attention.

L'utilisation de codage relatif des références au sein des micro-triangulations ou des sous-triangulations est un bon moyen pour maximiser les gains présentés. Le partage de l'information géométrique entre les sommets partagés par des sous-triangulations voisines est aussi une piste à explorer afin de maximiser le gain en mémoire.

La décomposition en sous-triangulations, telle qu'elle est présentée ici, peut être à la base de travaux futurs pour améliorer le rendement des algorithmes à mémoires externes, permettant ainsi de minimiser davantage les défauts de pages.

L'extension en dimensions supérieures est aussi un sujet à développer. Les catalogues en dimension 3 ne sont pas faciles à définir, et nécessitent plus de travail, et présentent plus de cas de figures à considérer. Cependant, le principe reste inchangé, et le gain est toujours envisagé. La décomposition d'une tétraèdrisation en sous-structures dans le but d'utiliser des références locales n'est pas non plus trivial est peut faire l'objet de travaux futurs.

Bibliographie

Bibliographie

[Alex 01] A. Alexandrescu. *Modern C++ Design : Generic Programming and Design Patterns Applied*. Addison-Wesley Professional, February 2001.

[Alli 01a] P. Alliez and M. Desbrun. "Progressive Compression for Lossless Transmission of Triangle Meshes". In : *SIGGRAPH '01 : Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 195–202, ACM Press, New York, NY, USA, 2001.

[Alli 01b] P. Alliez and M. Desbrun. "Valence-Driven Connectivity Encoding for 3D Meshes". *Computer Graphics Forum*, Vol. 20, No. 3, 2001.

[Alli 05] P. Alliez and C. Gotsman. "Recent Advances in Compression of 3D Meshes". In : N. A. Dodgson, M. S. Floater, and M. A. Sabin, Eds., *Advances in Multiresolution for Geometric Modelling*, pp. 3–26, Springer, Berlin, Heidelberg, 2005.

[Arge 04] L. Arge, G. S. Bordal, and R. Fagerberg. *Handbook Of Data Structures And Applications*, Chap. Cache-Oblivious Data Structures. *Chapman & Hall/Crc Computer and Information Science*, Chapman & Hall/CRC, 2004.

[Atte 03] M. Attene, B. Falcidieno, M. Spagnuolo, and J. Rossignac. "SwingWrapper : Retiling Triangle Meshes for Better Edgebreaker Compression". *ACM Trans. Graph.*, Vol. 22, No. 4, pp. 982–996, 2003.

[Baja 99] C. L. Bajaj, V. Pascucci, and G. Zhuang. "Single Resolution Compression of Arbitrary Triangular Meshes with Properties". In : *DCC '99 : Proceedings of the Conference on Data Compression*, p. 247, IEEE Computer Society, Washington, DC, USA, 1999.

[Baum 72] B. G. Baumgart. "Winged Edge Polyhedron Representation". Tech. Rep., Stanford University, Stanford, CA, USA, 1972.

[Baum 74] B. G. Baumgart. *Geometric Modeling for Computer Vision*. PhD thesis, Stanford University, USA, August 1974.

[Baum 75] B. G. Baumgart. "Winged-Edge Polyhedron Representation for Computer Vision". In : *National Computer Conference*, May 1975.

[Berg 00] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry Algorithms and Applications*. Springer-Verlag Berlin Heidelberg, New York, USA, 2000.

[Blan 03a] D. K. Blandford, G. E. Blelloch, D. E. Cardoze, and C. Kadow. "Compact Representations of Simplicial Meshes in Two and Three Dimensions". In : *the 12th International Meshing Roundtable*, pp. 135–146, 2003.

[Blan 03b] D. K. Blandford, G. E. Blelloch, and I. A. Kash. "Compact representations of separable graphs". In : *the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 679–688, January 12-14 2003.

[Blan 04] D. K. Blandford, G. E. Blelloch, and I. A. Kash. "An Experimental Analysis of a Compact Graph Representation". In : *ALENEX/ANALC*, pp. 49–61, 2004.

[Blan 05] D. K. Blandford, G. E. Blelloch, D. E. Cardoze, and C. Kadow. "Compact Representations of Simplicial Meshes in Two and Three Dimensions". *International Journal of Computatinal Geometry and Applications*, Vol. 15, No. 1, pp. 3–24, 2005.

[Blan 06] D. K. Blandford. *Compact Data Structures with Fast Queries*. PhD thesis, Carnegie Mellon University, USA, February 2006.

[Bois 02] J.-D. Boissonnat, O. Devillers, S. Pion, M. Teillaud, and M. Yvinec. "Triangulations in CGAL". *Computational Geometry Theory and Applications*, Vol. 22, No. 1-3, pp. 5–19, May 2002.

[Bois 95] J.-D. Boissonnat and M. Yvinec. *Géométrie algorithmique*. Ediscience international, Paris, 1995.

[Bond 76] J. A. Bondy. *Graph Theory With Applications*. Elsevier Science Ltd, 1976.

[Bose 97] P. Bose and G. Toussaint. "Characterizing and Efficiently Computing Quadrangulations of Planar Point Sets". *Computer Aided Geomtric Design*, Vol. 14, No. 8, pp. 763–785, 1997.

[Brey 97] U. Breymann. *Designing Components with the C++ STL : A New Approach to Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[Camp 98] S. Campagna, L. Kobbelt, and H.-P. Seidel. "Directed edges A scalable representation for triangle meshes". *Journal of Graphic Tools*, Vol. 3, No. 4, pp. 1–11, 1998.

[Cast 04] L. Castelli Aleardi, O. Devillers, and G. Schaeffer. "Compact Representation of Triangulations". Internal Report 5433, INRIA, 2004.

[Cast 05] L. Castelli Aleardi, O. Devillers, and G. Schaeffer. "Succinct Representation of Triangulations with a Boundary". In : *Proceedings of Workshop on Algorithms And Data Structures (WADS)*, pp. 134–145, 2005.

[Cast 06a] L. Castelli Aleardi. *Représentations Compactes de Structures de Données Géométriques*. PhD thesis, Ecole Polytechnique, Palaiseau, France, December 2006.

[Cast 06b] L. Castelli Aleardi, O. Devillers, and A. Mebarki. "2D Triangulation Representation Using Stable Catalogs". In : *Proceedings of the 18th Canadian Conference on Computational Geometry*, pp. 71–74, 2006.

[Cast 06c] L. Castelli Aleardi, O. Devillers, and G. Schaeffer. "Optimal Succinct Representations of Planar Maps". In : *SCG 06 : Proceedings of the 22nd ACM Symposium on Computational Geometry, Sedona, Arizona, USA, June 57, 2006*, pp. 309–318, ACM Press, New York, NY, USA, 2006.

[Cast 07] L. Castelli Aleardi, O. Devillers, and A. Mebarki. "Catalog-Based Representation of 2D Triangulations". Submitted to the International Journal on Computational Geometry and Applications, 2007.

[CGAL] "CGAL : Computational Geometry Algorithms Library. www.cgal.org".

[CGAL 07] CGAL Editorial Board. *CGAL User and Reference Manual*. 3.3 Ed., 2007.

[Chen 96] J. Chen. "Computational Geometry : Methods and Applications". February 1996. Manuscipt. Computer Science Department, Texas A&M University.

[Chia 01] Y.-T. Chiang, C.-C. Lin, and H.-I. Lu. "Orderly Spanning Trees with Applications to Graph Encoding and Graph Drawing". In : *SODA '01 : Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pp. 506–515, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[Chou 02] P. H. Chou and T. H. Meng. "Vertex Data Compression through Vector Quantization". *IEEE Transactions on Visualization and Computer Graphics*, Vol. 8, No. 4, pp. 373–382, 2002.

[Chow 97] M. M. Chow. "Optimized Geometry Compression for Real-Time Rendering". In : *VIS '97 : Proceedings of the 8th conference on Visualization '97*, pp. 347–ff., IEEE Computer Society Press, Los Alamitos, CA, USA, 1997.

[Cign 03] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. "External Memory Management and Simplification of Huge Meshes". *IEEE Transactions on Visualization and Computer Graphics*, Vol. 9, No. 4, pp. 525–537, 2003.

[Clin 84] A. K. Cline and R. J. Renka. "A Storage-efficient Method for Construction of a Thiessen Triangulation". *Rocky Mountain Journal of Mathematics*, Vol. 14, No. 1, pp. 119–139, 1984.

[Cohe 99] D. Cohen-Or, D. Levin, and O. Remez. "Progressive Compression of Arbitrary Triangular Meshes". In : *VIS '99 : Proceedings of the conference on Visualization '99*, pp. 67–72, IEEE Computer Society Press, Los Alamitos, CA, USA, 1999.

[Corm 01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw-Hill Book Company, 2001.

[Cove 06] T. M. Cover and J. A. Thomas. *Elements of Information Theory. Wiley Series in Telecommunications and Signal Processing*, Wiley-Interscience, 2006.

[Cran 06] K. Crane. "A Survey of Efficient Structures for Digital Geometry Processing". A survey written for Michael Garland's digital geometry processing class (in Department of Computer Science at the University of Illinois), April 2006. California Institute of Technology.

[Deer 95] M. Deering. "Geometry compression". In : *SIGGRAPH '95 : Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 13–20, ACM Press, New York, NY, USA, 1995.

[Dela 07] C. Delage. *Spatial Sorting, CGAL User and Reference Manual*. 3.3 Ed., 2007.

[Deo 04] N. Deo. *Handbook Of Data Structures And Applications*, Chap. Fundamentals : Graphs. *Chapman & Hall/Crc Computer and Information Science*, Chapman & Hall/CRC, 2004.

[Devi 01] O. Devillers, S. Pion, and M. Teillaud. "Walking in a Triangulation". In : *SCG '01 : Proceedings of the seventeenth annual symposium on Computational geometry*, pp. 106–114, ACM, New York, NY, USA, 2001.

[Dies 00] R. Diestel. *Graph Theory*. Vol. 173 of *Graduate Texts in Mathematics*, Springer-Verlag, New York, USA, 2000.

[EDan 05] E.Danovaro, L. Floriani, P. Magillo, and N.Sokolovky. "Encoding Level-Of-Detail Tetrahedral Meshes". In : N. A. Dodgson, M. S. Floater, and M. A. Sabin, Eds., *Advances in Multiresolution for Geometric Modelling*, pp. 89–100, Springer, Berlin, Heidelberg, 2005.

[Elia 75] P. Elias. "Universal Codeword Sets and Representations of the Integers". *Information Theory, IEEE Transactions on*, Vol. 21, No. 2, pp. 194–203, 1975.

[Fabr 00] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. "On the Design of CGAL a Computational Geometry Algorithms Library". *Software - Practice and Experience*, Vol. 30, No. 11, pp. 1167–1202, September 2000.

[Flor 05] L. D. Floriani, L. Kobbelt, and E. Puppo. "A Survey on Data Structures for Level-Of-Detail Models". In : N. A. Dodgson, M. S. Floater, and M. A. Sabin, Eds., *Advances in Multiresolution for Geometric Modelling*, pp. 49–74, Springer, Berlin, Heidelberg, 2005.

[Frey 00] P. J. Frey and P.-L. George. *Mesh Generation : Application to Finite Elements*. Kogan Page, May 2000.

[Gand 01] P. Gandoin. *Compression progressive et sans perte de structures géométriques*. PhD thesis, Université de Nice Sophia Antipolis, France, Septembre 2001.

[Gand 02] P. Gandoin and O. Devillers. "Progressive Lossless Compression of Arbitrary Simplicial Complexes". In : *SIGGRAPH '02 : Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pp. 372–379, ACM, New York, NY, USA, 2002.

[Garl 05] M. Garland. "A Multiresolution Representation for Massive Meshes". *IEEE Transactions on Visualization and Computer Graphics*, Vol. 11, No. 2, pp. 139–148, 2005. Student Member-Eric Shaffer.

[Gers 91] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[Good 00] M. T. Goodrich and K. Ramaiyer. *Handbook of Computational Geometry*, Chap. Geometric Data Structures, pp. 463–489. North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, 2000.

[Gots 01] C. Gotsman, S. Gumhold, and L. Kobbelt. "Simplification and compression of 3D meshes". In : *Proceedings of the European Summer School on Principles of Multiresolution in Geometric Modelling (PRIMUS)*, pp. 319–361, August 2001.

[Grif 76] H. B. Griffiths. *Surfaces*. Cambridge University Press, 1976.

[Guib 83] L. J. Guibas and J. Stolfi. "Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams". In : *STOC '83 : Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pp. 221–234, ACM Press, New York, NY, USA, 1983.

[Gumh 00] S. Gumhold. *Mesh Compression*. PhD thesis, University of Tübingen, July 2000.

[Gumh 98] S. Gumhold and W. Straßer. "Real Time Compression of Triangle Mesh Connectivity". In : *SIGGRAPH '98 : Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 133–140, ACM Press, New York, NY, USA, 1998.

[Halb 99] Y. Halbwachs and Ø. Hjelle. *Advances in Software Tools for Scientific Computing*, Chap. Generalized maps in geological modeling : Object-oriented design of topological kernels, pp. 339–356. Vol. 10 of *Lecture Notes in Computational Science and Engineering*, Langtangen, Hans P. ; Bruaset, Are M. ; Quak, Ewald (Eds.). Springer-Verlag, editors, 1999.

[He 99] X. He, M.-Y. Kao, and H.-I. Lu. "Linear-Time Succinct Encodings of Planar Graphs via Canonical Orderings". *SIAM J. Discret. Math.*, Vol. 12, No. 3, pp. 317–325, 1999.

[Heig 83] E. Heighway. "A Mesh Generator for Automatically Subdividing Irregular Polygons into Quadrilaterals". *IEEE Transactions on Magnetics*, Vol. 19, No. 6, pp. 2535–2538, November 1983.

[Hjel 00] Ø. Hjelle. "A Triangulation Template Library (TTL) : Generic Design of Triangulation Software". Tech. Rep. STF42 A00015, SINTEF Applied Mathematics, Oslo, 2000.

[Hjel 06] Ø. Hjelle and M. Dæhlen. *Triangulations and Applications (Mathematics and Visualization)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Hopp 96] H. Hoppe. "Progressive Meshes". In : *SIGGRAPH '96 : Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 99–108, ACM Press, New York, NY, USA, 1996.

[Isen] M. Isenburg. *Compression and Streaming of Polygon Meshes*. PhD thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA.

[Isen 00a] M. Isenburg. "Triangle Fixer : Edge-based Connectivity Compression". In : *EWCG'00 : Proceedings of the 16th European Workshop on Computational Geometry*, pp. 18–23, 2000.

[Isen 00b] M. Isenburg and J. Snoeyink. "Face Fixer : Compressing Polygon Meshes with Properties". In : *SIGGRAPH '00 : Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 263–270, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[Isen 02a] M. Isenburg. "Compressing Polygon Mesh Connectivity with Degree Duality Prediction". In : *Proceedings of Graphics Interface 2002, Calgary, Alberta, Canada, 27-29 May 2002*, pp. 161–170, 2002.

[Isen 02b] M. Isenburg and P. Alliez. "Compressing Hexahedral Volume Meshes". In : *PG '02 : Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, p. 284, IEEE Computer Society, Washington, DC, USA, 2002.

[Isen 02c] M. Isenburg and P. Alliez. "Compressing Polygon Mesh Geometry with Parallelogram Prediction". In : *VIS '02 : Proceedings of the conference on Visualization '02*, IEEE Computer Society, Washington, DC, USA, 2002.

[Isen 03a] M. Isenburg and P. Alliez. "Compressing Hexahedral Volume Meshes". *Graph. Models*, Vol. 65, No. 4, pp. 239–257, 2003.

[Isen 03b] M. Isenburg and S. Gumhold. "Out-of-Core Compression for Gigantic Polygon Meshes". *ACM Trans. Graph.*, Vol. 22, No. 3, pp. 935–942, 2003.

[Isen 03c] M. Isenburg, P. Lindstrom, S. Gumhold, and J. Snoeyink. "Large Mesh Simplification using Processing Sequences". In : *VIS '03 : Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, p. 61, IEEE Computer Society, Washington, DC, USA, 2003.

[Isen 05a] M. Isenburg and P. Lindstrom. "Streaming Meshes". In : *Proceedings of the 16th IEEE Visualization Conference (VIS 2005), 23-28 October 2005, Minneapolis, MN, USA*, p. 30, IEEE Computer Society, 2005.

[Isen 05b] M. Isenburg, P. Lindstrom, and J. Snoeyink. "Streaming Compression of Triangle Meshes". In : *SGP '05 : Proceedings of the third Eurographics symposium on Geometry processing*, p. 111, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2005.

[Isen 05c] M. Isenburg and J. Snoeyink. "Graph Coding and Connectivity Compression". In : *Proceedings of the China-Japan Joint Conference on Discrete Geometry, Combinatorics and Graph Theory (CJCDGCGT) Nankai University, Tianjin, China ; Northwestern Polytechnical University, Xi'an, China and Tokai University, Japan*, November 18-24 2005.

[Isen 06a] M. Isenburg, P. Lindstrom, S. Gumhold, and J. Shewchuk. "Streaming Compression of Tetrahedral Volume Meshes". In : *GI '06 : Proceedings of Graphics Interface 2006*, pp. 115–121, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 2006.

[Isen 06b] M. Isenburg, Y. Liu, J. Shewchuk, and J. Snoeyink. "Streaming Computation of Delaunay Triangulations". In : *SIGGRAPH '06 : ACM SIGGRAPH 2006 Papers*, pp. 1049–1056, ACM Press, New York, NY, USA, 2006.

[ISOIa] ISO/IEC 14772-2 :2004. "Virtual Reality Modeling Language (VRML)". International Organisation for Standardisation, Geneva, Switzerland.

[ISOIb] ISO/IEC 14882 :2003. "Programming Languages c++". International Organisation for Standardisation, Geneva, Switzerland.

[Kall 01a] M. Kallmann. *Object Interaction in Real-Time Virtual Environments*. PhD thesis, Swiss Federal Institute of Technology (EPFL), January 2001. thesis number 2347.

[Kall 01b] M. Kallmann and D. Thalmann. “Star Vertices : A Compact Representation for Planar Meshes with Adjacency Information”. *Journal of Graphics Tools*, Vol. 6, No. 1, pp. 7–18, 2001.

[Keel 95] K. Keeler and J. Westbrook. “Short Encodings of Planar Graphs and Maps”. *Discrete Appl. Math.*, Vol. 58, No. 3, pp. 239–252, 1995.

[Kett 07] L. Kettner. *Halfedge Data Structures, CGAL User and Reference Manual*. 3.3 Ed., 2007.

[Kett 99] L. Kettner. “Using Generic Programming for Designing a Data Structure for Polyhedral Surfaces”. *Computational Geometry Theory and Applications*, Vol. 13, No. 1, pp. 65–90, 1999. Elsevier Science Publishers B. V. Amsterdam, The Netherlands.

[Khod 00] A. Khodakovsky, P. Schröder, and W. Sweldens. “Progressive Geometry Compression”. In : *SIGGRAPH ’00 : Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 271–278, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[Khod 02] A. Khodakovsky, P. Alliez, M. Desbrun, and P. Schröder. “Near-Optimal Connectivity Encoding of 2-Manifold Polygon Meshes”. *Graph. Models*, Vol. 64, No. 3/4, pp. 147–168, 2002.

[King 99] D. King and J. Rossignac. “Guaranteed 3.67V bit Encoding of Planar Triangle Graphs”. In : *Proceedings of the Canadian Conference on Computational Geometry*, 1999.

[Kron 01] B. Kronrod and C. Gotsman. “Efficient Coding of Nontriangular Mesh Connectivity”. *Graph. Models*, Vol. 63, No. 4, pp. 263–275, 2001.

[Lasz 95] M. J. Laszlo. *Computational Geometry and Computer Graphics in C++*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[Lavo 05] G. Lavoué. *Compresson de surfaces, basée sur la subdivision inverse, pour la transmission bas débit et la visualisation progressive*. PhD thesis, Université Claude Bernard Lyon 1, France, December 2005.

[Lee 00] E.-S. Lee and H.-S. Ko. “Vertex Data Compression for Triangular Meshes”. In : *PG ’00 : Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, p. 225, IEEE Computer Society, Washington, DC, USA, 2000.

[Lee 02] H. Lee, P. Alliez, and M. Desbrun. “Angle-Analyzer : A Triangle-Quad Mesh Codec”. *Computer Graphics Forum*, Vol. 21, No. 3, pp. 383–383, 2002.

[Lewi] T. Lewiner. *Mesh Compression from Geometry.* PhD thesis, Université de Pierre et Marie Curie (Paris VI), Paris.

[Lewi 04] T. Lewiner, H. Lopes, J. Rossignac, and A. W. Vieira. "Efficient Edgebreaker for Surfaces of Arbitrary Topology". In : *SIBGRAPI '04 : Proceedings of the Computer Graphics and Image Processing, XVII Brazilian Symposium on (SIBGRAPI'04)*, pp. 218–225, IEEE Computer Society, Washington, DC, USA, 2004.

[Li 98] J. Li and C.-C. Jay Kuo. "A Dual Graph Approach to 3D Triangular Mesh Compression". In : *ICIP 98 : Proceedings of International Conference on Image Processing, 4-7 Oct 1998, Chicago, IL, USA*, pp. 891–894, October 1998.

[Lien 89] P. Lienhardt. "Subdivisions of N-Dimensional Spaces and N-Dimensional Generalized Maps". In : *SCG '89 : Proceedings of the Fifth Annual Symposium on Computational geometry*, pp. 228–236, ACM Press, New York, NY, USA, 1989.

[Lind 00] P. Lindstrom. "Out-of-Core Simplification of Large Polygonal Models". In : *SIGGRAPH '00 : Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 259–262, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[Lind 01] P. Lindstrom and C. T. Silva. "A Memory Insensitive Technique for Large Model Simplification". In : *VIS '01 : Proceedings of the conference on Visualization '01*, pp. 121–126, IEEE Computer Society, Washington, DC, USA, 2001.

[Lope 02] H. Lopes, G. Tavares, J. Rossignac, A. Szymczak, and A. Safanova. "Edgebreaker : a Simple Compression for Surfaces with Handles". In : *SMA '02 : Proceedings of the seventh ACM symposium on Solid modeling and applications*, pp. 289–296, ACM Press, New York, NY, USA, 2002.

[Mant 87] M. Mäntylä. *An Introduction to Solid Modeling.* Computer Science Press, Inc., New York, NY, USA, 1987.

[Meht 04] D. P. Mehta and S. Sahni. *Handbook Of Data Structures And Applications. Chapman & Hall/Crc Computer and Information Science*, Chapman & Hall/CRC, 2004.

[Mull 78] D. E. Muller and F. P. Preparata. "Finding the Intersection of two Convex Polyhedra". *Theor. Comput. Sci.*, Vol. 7, pp. 217–236, 1978.

[Muss 95] D. R. Musser and A. Saini. *The STL Tutorial and Reference Guide : C++ Programming with the Standard Template Library.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1995.

[Paja 00] R. Pajarola and J. Rossignac. "Compressed Progressive Meshes". *IEEE Transactions on Visualization and Computer Graphics*, Vol. 6, No. 1, pp. 79–93, 2000.

[Peng 05] J. Peng, C.-S. Kim, and C. C. J. Kuo. "Technologies for 3D Mesh Compression : a Survey". *J. Visual Communication and Image Representation*, Vol. 16, No. 6, pp. 688–733, 2005.

[Pete 91] J. Petersen. "Die Theorie der regularen Graphs (The theory of regular graphs)". *Acta Mathematica*, Vol. 15, pp. 193–220, 1891.

[Pion 99] S. Pion. *De la géométrie algorithmique au calcul géométrique*. PhD thesis, Université de Nice Sophia Antipolis, France, 1999.

[Poul 03] D. Poulalhon and G. Schaeffer. "Optimal Coding and Sampling of Triangulations". In : *Proceedings of Thirtieth International Colloquium on Automata, Languages and Programming*, pp. 1080–1094, 2003.

[Poul 06] D. Poulalhon and G. Schaeffer. "Optimal Coding and Sampling of Triangulations". *Algorithmica*, Vol. 46, No. 3, pp. 505–527, 2006.

[Prep 85] F. P. Preparata and M. I. Shamos. *Computational geometry : an introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[Rama 98] S. Ramaswami, P. Ramos, and G. Toussaint. "Converting Triangulations to Quadrangulations". *Computational Geometry Theory and Applications*, Vol. 9, No. 4, pp. 257–276, 1998.

[Ross 01] J. Rossignac. "3D Compression Made Simple : Edgebreaker with Zip&Wrap on a Corner-Table". In : *Proceedings of the International Conference on Shape Modeling and Applications (SMI 2001), 7-11 May 2001, Genoa, Italy*, pp. 278–, IEEE Computer Society, Washington, DC, USA, 2001.

[Ross 03a] J. Rossignac. "3D Mesh Compression". Tech. Rep. GIT-GVU-03-21, Georgia Institute of Technology, Atlanta, GA, USA, 2003.

[Ross 03b] J. Rossignac, A. Safonova, and A. Szymczak. *Hierarchical and Geometrical Methods in Scientific Visualization*, Chap. Edgebreaker on a Corner Table : A simple technique for representing and compressing triangulated surfaces, pp. 041–050. *Mathematics and Visualization*, Farin, Gerald ; Hamann, Bernd ; Hagen, Hans (Eds.) Springer-Verlag, 2003.

[Ross 04] J. Rossignac. *Visualization Handbook*, Chap. 3D Mesh Compression, pp. 339–356. Vol. 10, Academic Press, Inc., Orlando, FL, USA, 2004.

[Ross 99a] J. Rossignac. "Edgebreaker : Connectivity Compression for Triangle Meshes". *IEEE Transactions on Visualization and Computer Graphics*, Vol. 5, No. 1, pp. 47–61, 1999.

[Ross 99b] J. Rossignac and A. Szymczak. "WRAP&Zip Decompression of the Connectivity of Triangle Meshes Compressed with Edgebreaker". *Comput. Geom. Theory Appl.*, Vol. 14, No. 1-3, pp. 119–135, 1999.

[Same 90a] H. Samet. *Applications of Spatial Data Structures : Computer Graphics, Image processing, and GIS*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[Same 90b] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[Schn 90] W. Schnyder. "Embedding Planar Graphs on the Grid". In : *SODA '90 : Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pp. 138–148, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1990.

[Sedg 84] R. Sedgewick. *Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[Shew 96] J. R. Shewchuk. "Triangle : Engineering a 2D Quality Mesh Generator and Delaunay Triangulator". In : M. C. Lin and D. Manocha, Eds., *Applied Computational Geometry : Towards Geometric Engineering*, pp. 203–222, Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.

[Silv] C. Silva, Y. Chiang, J. El-Sana, and P. Lindstrom. "Out-of-Core Algorithms for Scientific Visualization and Computer Graphics". In IEEE Visualization conference'02. Boston, Massachusets. Course Notes, October, 2002.

[Stan 05] *Standard Template Library Programmer's Guide*. SGI, 2005.

[Szym 00] A. Szymczak and J. Rossignac. "Grow&Fold : Compressing the Connectivity of Tetrahedral Meshes". 2000".

[Szym 01] A. Szymczak, D. King, and J. Rossignac. "An Edgebreaker-Based Efficient Compression Scheme for Regular Meshes". *Comput. Geom. Theory Appl.*, Vol. 20, No. 1-2, pp. 53–68, 2001.

[Taub 98] G. Taubin and J. Rossignac. "Geometric Compression through Topological Surgery". *ACM Trans. Graph.*, Vol. 17, No. 2, pp. 84–115, 1998.

[Toum 98] C. Touma and C. Gotsman. "Triangle Mesh Compression". In : *Proceedings of the Graphics Interface 1998 Conference, June 18-20, 1998, Vancouver, BC, Canada*, pp. 26–34, Canadian Human-Computer Communications Society, 1998.

[Tous 95] G. T. Toussaint. "Quadrangulations of Planar Sets". pp. 218–227, 1995.

[TRIANGLE] "TRIANGLE : Mesh Generator and Delaunay Triangulator".

[Tura 84] G. Turan. “Succinct Representations of Graphs”. *Discrete Applied Mathematics*, Vol. 8, pp. 289–294, 1984.

[Tutt 62] W. Tutte. “A Census of Planar Triangulations”. *Canadian Journal of Mathematics*, Vol. 14, pp. 21–38, 1962.

[Vand 02] D. Vandevoorde and N. M. Josuttis. *C++ Templates : The Complete Guide*. Addison Wesley, November 12 2002.

[Vitt 99] J. S. Vitter. “External Memory Algorithms and Data Structures”. pp. 1–38, 1999.

[Weil 85] K. Weiler. “Edge-based Data Structures for Solid Modeling in Curved-Surface Environments”. *IEEE Computer graphics and applications*, Vol. 5, No. 1, pp. 21–40, 1985.

[Wu 03] J. Wu and L. Kobbelt. “A Stream Algorithm for the Decimation of Massive Meshes”. In : *Graphics Interface*, pp. 185–192, CIPS, Canadian Human-Computer Commnication Society, A K Peters, June 2003. ISBN 1-56881-207-8, ISSN 0713-5424.

[Zach 03] G. Zachmann and E. Langetepe. “Geometric Data Structures for Computer Graphics”. In : *Proceedings of ACM SIGGRAPH*, ACM Transactions of Graphics, 27–31 July 2003.

Index

Index