M2R Image Vision et Robotique 2005-2006

Sébastien *BARBIER*

Exploration interactive de gros maillages surfaciques et volumiques

Encadrant
Georges-Pierre *BONNEAU*

Co-Encadrant
Lionel *REVÉRET*

Laboratoire : GRAVIR
Équipe : EVASION

Table des matières

Introduction

1. Contexte

La visualisation de données scientifiques maillées connait depuis une décennie de nombreux bouleversements liés à l'évolution des outils d'acquisition et de simulation de données.

Les moyens d'acquisition de points sur des objets physiques ont permis la création de maillages surfaciques de plus en plus complexes en nombre de triangles, motivant la recherche d'algorithmes de simplification diminuant le nombre de ces triangles afin de permettre simultanément une visualisation interactive et un stockage en mémoire vive de ces grands modèles de données. La mise en place récente du projet `Digital Michelangelo Project` fournissant des modèles surfaciques composés de millions de sommets et donc de triangles, a stimulé la création de nouveaux algorithmes incluant des traitements en mémoire externe sans lesquels toute visualisation serait impossible.

Dans la même optique, l'utilisation de maillages très fins et de nombreux pas de temps dans le cadre de simulation dynamique peut aboutir à des masses de données de l'ordre du tera ou du peta octets. Ces ensembles de données massives proviennent des centres de calculs tels que le `CEA` ou d'`EDF R&D`. Là encore, des méthodes adaptées de visualisation sont indispensables.

2. Objectifs

Le sujet de ce stage sur l'exploration interactive de gros maillages surfaciques ou volumiques s'inscrit dans la lignée de ces algorithmes de visualisation.

Nous cherchons en effet à visualiser de gros maillages de plusieurs millions de sommets que nous aurons préalablement simplifiés afin de pouvoir les afficher rapidement. Une région d'intérêt de focalisation dont la forme et le centre seront définis par l'utilisateur et pouvant être déplacée, affiche en son sein un maillage fin et précis afin de montrer les détails de l'objet en cet endroit. En dehors de cette région, l'objet reste à sa résolution la plus faible, c'est-à-dire sous forme d'un maillage grossier.

L'originalité de ce sujet est de créer une application permettant une visualisation **interactive** reposant sur des algorithmes astucieux et en complexité la plus petite possible. Elle est de plus générale, c'est-à-dire aussi efficace sur des maillages surfaciques que volumiques.

Notre méthode permet de gérer des maillages de taille arbitraire grâce à des algorithmes en mémoire externe.

De plus, il est possible de déporter la visualisation de la région d'intérêt sur une station distante, en communiquant un minimum d'informations entre la machine gérant le maillage entier et la station affichant la zone d'intérêt.

Chapitre 1

État de l'art

Le domaine de la simplification et de la visualisation de gros maillages de données dans lequel s'inscrit ce stage a connu un essor conséquent lors de la dernière decennie. De nombreux algorithmes résultent de ces années de recherche permettant de réaliser des simplifications de maillages puis ensuite de les visualiser ou de les stocker de manière efficace.

Ce chapitre cherche donc à présenter de manière simple et structurée les différents algorithmes existants afin d'établir les raisons de notre approche. Nous reviendrons tout d'abord plus précisément sur les méthodes de simplification de maillages permettant de créer ce que nous appelons un maillage grossier (par rapport au maillage initial fin) dont ceux basés sur la notion de clusterisation, point-clé de notre algorithme. Ensuite, nous détaillerons les algorithmes de visualisation surfacique et volumique basés sur la notion de multirésolution en faisant la distinction entre algorithme *in-core* et algorithme *out-of-core*.

De plus, de nombreux articles tentent de regrouper l'ensemble de ces algorithmes parfois sur la base de critères particuliers. On peut ainsi par exemple se référer à celui de Heckbert [HG97], de Luebke [Lue01] ou de Garland [Gar99] pour les algorithmes de simplification et multirésolution ou encore plus récemment à celui de De Floriani [DFKP04] pour les structures de données multirésolution. Ces articles sont beaucoup plus généraux et dressent sous la forme d'un catalogue un horizon complet des techniques existantes.

1.1 Méthodes de simplification et clusterisation

Nous présentons tout d'abord les principales méthodes de simplification. Ces méthodes créent, à partir d'un maillage fin en entrée, un maillage dit grossier composé d'un nombre inférieur de sommets par rapport à l'original. Le maillage ainsi obtenu occupera une place mémoire moins importante et pourra être affiché plus rapidement dû à ce nombre moindre de triangles. De plus, nous nous attarderons dans cette partie plus longuement sur la méthode de clusterisation qui est utilisée par notre algorithme.

1.1.1 Méthodes de simplification

Dans cette partie, nous détaillons trois types d'algorithmes : des méthodes de décimation, tout d'abord, consistant à effondrer des arêtes afin de fusionner des sommets ; puis un algorithme de retriangulation basé sur la suppression de triangles autour d'un sommet ; et enfin, un algorithme regroupant les faces du maillage par ensemble pour créer de plus grandes faces.

D'autres méthodes existent comme le Re-Tiling [Tur92] ou le Volume Rendering [HHK+95] mais nous nous intéressons ici à des algorithmes simples qui agissent directement sur la définition du maillage initial.

Edge Collapse et Vertex Pair Collapse

Le premier type d'algorithmes dit de décimation, permet d'effondrer selon différents critères un certain nombre d'arêtes. Chaque effondrement va enlever au maillage initial un certain nombre de sommets, d'arêtes et de triangles. Il existe différentes versions de décimation mais les plus utilisées restent les effondrements d'arêtes.

La notion d'effondrement d'arêtes ou *Full Edge Collapse* a été introduite par Hoppe [HDD+93]. Le principe est le suivant. On choisit, grâce à des critères d'erreurs souvent placés dans l'espace-écran et en utilisant une notion de distance ou d'énergie que l'on cherche à minimiser, une arête dans le maillage que l'on va effondrer sur elle-même pour ainsi fusionner ses deux sommets extrémités ensemble et enlever les deux triangles voisins de l'arête. Le choix de la localisation du nouveau sommet résultant dépend ensuite de la méthode utilisée. Il se trouve souvent sur l'un des deux anciens sommets (*Half Edge Collapse* [Hop96]) ou au milieu.

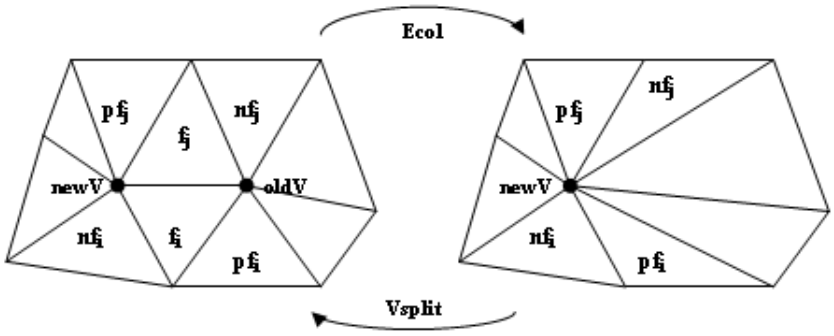
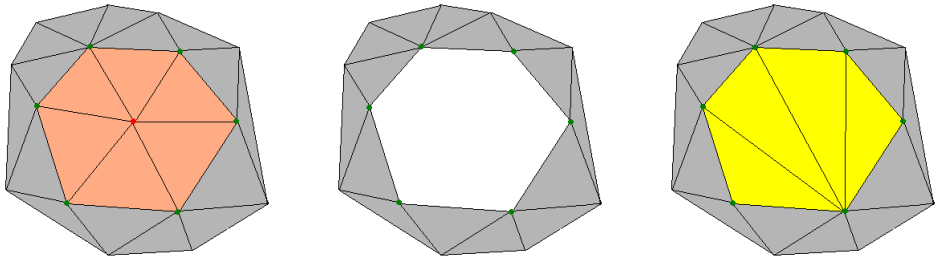


Figure 1 : Half Edge Collapse.
Opérations d'effondrement (**E**dge **Col**lapse) et de raffinement (**V**ertex **Split**)

Une autre méthode plus générale proposée par Garland [GH97] et qu'il a nommée *Vertex Pair Collapse*, consiste à choisir parmi l'ensemble des sommets, certaines paires, puis à les effondrer. Le choix de ces sommets est déterminant car il impose la complexité de l'algorithme.

Vertex Removal

Le second type d'algorithmes n'enlève plus des arêtes comme précédemment mais directement des sommets. Ces sommets sont choisis grâce à une mesure métrique qui minimise la différence entre le maillage final grossier et le maillage fin initial. Cette méthode proposée par Schroeder [SZL92] fonctionne de la manière suivante. Lorsqu'un sommet est choisi, l'ensemble de ses triangles adjacents sont enlevés laissant un "trou". Ce trou est ensuite comblé par une triangulation adaptée obtenue par projection sur un plan.



Choix du sommet Création du trou Retriangulation

Figure 2 : Vertex Removal
Un sommet est choisi. Les triangles fins l'ayant pour sommet sont enlevés. Le trou ainsi formé est retriangulé (en un nombre moins important de triangles).

Face Merging

Ce dernier type d'algorithmes proposé tout d'abord par De Haemer [DHZ91] puis ensuite amélioré par la méthode des *superfaces* [KT96], rassemble des faces du maillage initial par un test de coplanarité afin de créer un maillage grossier. Plus précisement, l'algorithme unit un ensemble de faces dont les normales respectives sont très proches. Une fois, ces grandes faces déterminées, leurs arêtes sont calculées et sous certains critères peuvent être raffinées. Enfin, les grandes faces sont retriangulées.

1.1.2 Clusterisation

La clusterisation consiste à regrouper les sommets du maillage fin initial en un nombre fini d'ensembles. Chaque ensemble déterminera alors un sommet grossier. Nous détaillerons dans la suite plus précisement comment obtenir un maillage grossier. Mais tout d'abord, nous présentons plusieurs méthodes permettant de réaliser une clusterisation sur des maillages.

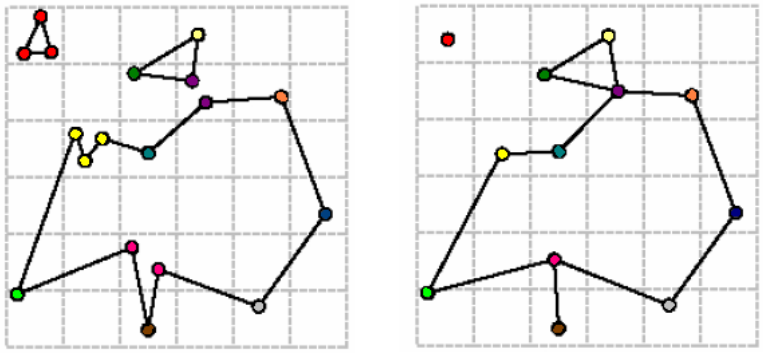
Précisons que cette méthode est, depuis l'arrivée des gros maillages de données (comme ceux issus du projet `Michelangelo` [LPC+00]), très utilisée car elle permet une simplification simple et rapide *en mémoire externe* de ceux-ci. La plupart des algorithmes cités ci-dessous ont donc été développés pour un traitement *en mémoire externe* de gros maillages de données.

La clusterisation est utile dans bien d'autres domaines. Un tour d'horizon plus général de l'ensemble des algorithmes permettant de la réaliser est disponible dans [Ber02].

Avec une grille régulière

La première méthode de clusterisation pour la simplification de maillages a été proposée par Rossignac et Borrel [RB93]. Elle se base sur la construction d'une grille régulière sur le maillage initial. Les sommets se trouvant tous dans le même voxel sont alors regroupés en un sommet grossier. Les arêtes et les triangles inclus dans un unique voxel sont effondrés. Tout simplexe ayant des sommets dans des voxels différents subi des effondrements pour des sommets associés au même voxel, les autres n'étant pas modifiés.

Le sommet représentatif de chaque cellule est choisi comme le barycentre des sommets fins pour cette méthode. Lindstrom [Lin00] et Shaffer [SG01] proposent une autre méthode dans le cadre d'une simplification *en mémoire externe* utilisant l'erreur métrique quadrique définie dans [GH97]. Cela permet de minimiser la différence entre le maillage initial et le maillage grossier en choisissant sur des critères géométriques (distance euclidienne entre un point et un plan) la meilleure localisation pour le sommet grossier dans chaque voxel.



avant après

Figure 3 : Méthode de clusterisation avec une grille régulière.

Les deux méthodes suivantes tentent de pallier à l'inconvénient majeur de cette méthode, c'est-à-dire une simplification agressive pouvant créer des maillages dits *non-manifolds* en essayant de tenir compte des informations de connectivité locale.

Par contraction d'arêtes

Garland propose aussi une méthode de clusterisation basée sur la contraction d'arêtes [GWH01]. En considérant le maillage dual du maillage initial, il souligne la correspondance entre une contraction d'arêtes et la création d'un *cluster* composé de deux faces, les deux faces adjacentes à l'arête effondrée. En déroulant ainsi l'algorithme de contraction d'arêtes, il construit un ensemble fini de clusters sur les faces du maillage initial qui permettra de créer un maillage grossier.

Grille régulière et contraction d'arêtes

Garland concilie dans [GS02] les deux méthodes précédemment décrites dans le cadre d'un algorithme en mémoire externe en réalisant deux passes. La première en mémoire externe uti-

lise une grille régulière et une erreur quadrique métrique afin de simplifier grossièrement le maillage. La seconde effectue en mémoire vive des contractions d'arêtes par itération.

Garland propose ainsi une méthode qui utilise les avantages des deux méthodes. Une simplification efficace en temps et en mémoire est d'abord réalisée grâce à la grille. Le maillage ainsi simplifié, d'une taille donc raisonnable, peut-être chargé en mémoire vive pour subir des contractions d'arêtes qui prennent en considération la topologie du maillage.

METIS

`METIS` est un algorithme développé par Karypis et Kumar [KK98] afin de réaliser des *k-partitions* de graphe, c'est-à-dire constuire k ensembles indépendants dans un graphe. C'est donc une méthode très généraliste qui s'appuie sur l'effondrement d'arêtes dans un graphe, sur la création en parallèle de k ensembles dans ce graphe simplifié et qui ensuite, par reconstruction, détermine k ensembles dans le graphe initial.

On peut donc appliquer cet algorithme au maillage initial que l'on peut considérer comme un graphe. Celui-ci va renvoyer un fichier `metis` qui permettra par la suite de créer une clusterisation selon les sommets ou les faces du maillage initial.

Un des avantages majeurs de cet algorithme est sa rapidité grâce à une simplification des données d'entrée puis une parallélisation du traitement. De plus, il peut prendre en compte la géométrie du maillage par l'intermédiaire de poids attribués aux nœuds du graphe ; et surtout, il ne dépend d'aucune grille régulière extérieure au maillage, mais uniquement de données intrinsèques au maillage. Par contre, il ne fonctionne qu'en mémoire interne.

Avec un front courant : *Processing Sequence*

Cette méthode a été proposée simultanément par Isenburg et Lindstrom [ILGS03] d'un côté et Wu et Kobbelt [WK03] de l'autre, afin de réaliser des simplifications en mémoire externe ayant accès à tout instant à des informations géométriques et de connexité qu'une clusterisation classique ne permet pas d'avoir. Nous détaillons ici celle proposée par Isenburg et Lindstrom nommée *processing sequence*.

Une *processing sequence* est un maillage avec un ordre précis pour ses triangles et ses sommets.

Tout est réalisé grâce à deux *fronts de traitement* et la zone d'attente qu'ils délimitent. Le front dit de lecture ajoute des triangles dans la zone d'attente. Il est mis à jour en ajoutant un triangle à la fois en appliquant une des cinq règles possibles en fonction de la configuration. Le front dit d'écriture enlève les triangles de la zone qui ont leur trois sommets dans le front d'écriture.

Au sein de la zone d'attente, les triangles sont ordonnés. Dans cette zone, la connectivité entre triangles est donc connue. Ensuite, les triangles sont effondrés par rapport à une grille régulière et les relations de connexité. On peut donc avoir dans certains cas, plusieurs sommets grossiers dans une même case de la grille. En fait, chaque nouveau sommet fin se voit associé un sommet grossier. Si par la suite, une arête le relie à un autre sommet fin situé dans la même case alors les deux sommets grossiers associés sont fusionnés en un seul. Mais si une telle relation d'adjacence n'existe pas, les deux sommets grossiers ne seront jamais fusionnés et donc resteront dans la même case.

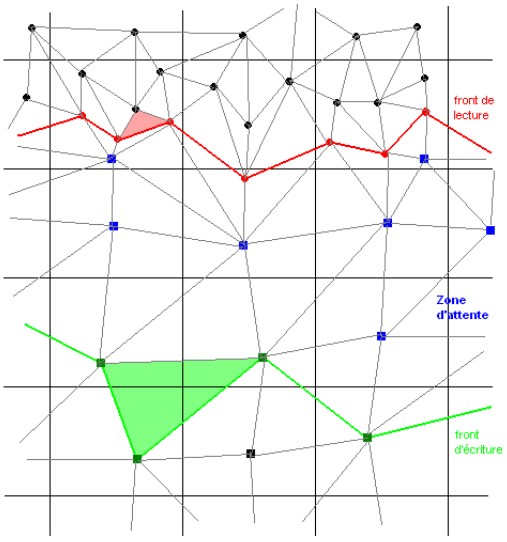


Figure 4 : Processing Sequence
Le triangle rouge sera le prochain à entrer dans le front de lecture alors que le triangle vert sortira de la zone d'attente via le front d'écriture.

1.2 Multirésolution et niveaux de détails en mémoire vive

La visualisation de gros maillages de données a été réalisée au cours de la dernière décennie en employant différents niveaux de détails (*Level-of-Details* ou LOD) du maillage définis par différents critères spécifiés par l'utilisateur. La visualisation est dite alors multirésolution car elle permet d'afficher une multitude de représentations du même maillage.

La visualisation multirésolution s'est développée en tentant de concilier trois attentes. La première est la définition de critères de simplification qui permettraient de conserver certains détails jugés important par l'utilisateur. La seconde est de stocker d'une manière efficace en mémoire les différents maillages afin de limiter leur place. La troisième est de garantir une interactivité de l'affichage pour le confort de l'utilisateur. Comme dans de nombreux domaines, l'occupation mémoire et la rapidité (ou complexité en temps) des algorithmes proposés nécessitent un bon compromis.

L'ensemble des algorithmes que nous allons détaillé ci-dessous repose sur les méthodes de simplification que nous avons détaillées auparavant et, dans cette section, est réalisé en mémoire vive. De plus, la majorité des algorithmes proposés jusqu'ici sont efficaces uniquement pour les maillages surfaciques même si certains travaux concernent des maillages volumiques comme ceux de De Floriani et al. avec `TAn2` [CDFM+02] ou ceux réalisés durant la thèse de Fabien Vivodztev [VBLT05].

1.2.1 Niveaux de détails uniformes

Un des premiers travaux sur la visualisation multirésolution a été proposé par Hugues Hoppe [Hop96]. Il proposait une série d'opérations réversibles d'*edge collapses* stockée de manière à pouvoir afficher différents niveaux de détails. On commence en fait avec un maillage simplifié grossier que l'on va préciser au fur et à mesure en réalisant des dédoublements de sommets (*vertex split*) dans l'ordre mémorisé. Hoppe nommera ce format des *progressive meshes*.

Mais les maillages obtenus par cette méthode étaient simplifiés de manière uniforme ce qui ne permettait pas de visualiser correctement les objets affichés.

Les algorithmes qui vont être développés dès lors, empêcheront la simplification de certaines parties du maillage en fonction de certains critères que nous allons détaillés, pour pallier à cet inconvénient et permettre ainsi une meilleure visualisation, c'est-à-dire aucune perte de détails visuels lors du rendu malgré la simplification.

1.2.2 Niveaux de détails sélectifs

Les critères

Les différents critères utilisés pour réaliser des niveaux de détails sélectifs sont de deux sortes : ceux basés sur la perception visuelle de l'utilisateur et ceux dépendant des propriétés intrinsèques du maillage triangulé.

Hoppe [Hop97] définit l'année suivant ses travaux sur les niveaux de détails uniformes trois critères basés sur la perception visuelle dont deux commums avec ceux proposés la même année par Xia et Varshney [XV96].

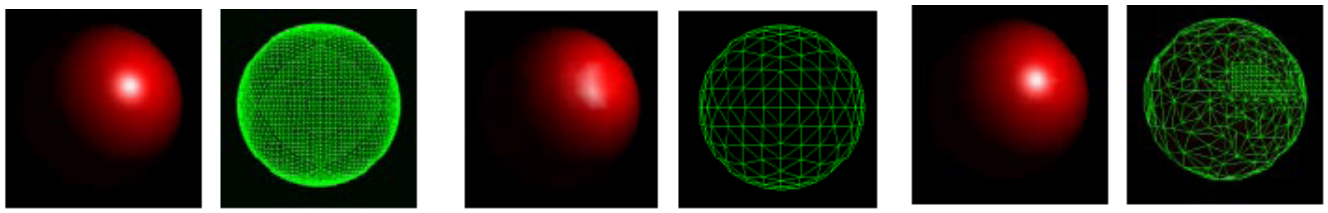
Le premier est la *view frustum* ou cône de vue : toute partie du maillage hors de celle-ci peut être simplifiée sans restriction.

Le deuxième est le *back face culling* ou critère de visibilité dans le cône de vue : toute partie située dans le cône de vue mais invisible au regard de l'utilisateur peut être grossière.

Enfin, le dernier est la distance à l'écran. Ce critère agit globalement sur le maillage et le raffine ou le simplifie en fonction de son éloignement par rapport à l'observateur. Plus un objet sera loin, plus il pourra être affiché à une résolution grossière.

Les autres critères de simplification sont plus liés aux propriétés "physiques" de l'objet. Hoppe, par exemple, prend en compte dans ces simplifications des champs scalaires (couleur par exemple...) pouvant être rattachés au maillage via les sommets. Xia et Varshney se préoccupe plus des conditions d'éclairage du maillage.

Avec l'apparition des maillages volumiques, on a aussi tenté de conserver d'autres propriétés importantes des maillages comme les matériaux [VBLT05] ou encore les isosurfaces [CDFM+02].



Maillage fin initial | Niveau de détails uniforme | Niveau de détails sélectif

Figure 5 : Différences entre niveaux de détails uniformes et sélectifs basés sur les critères : lumières, visibilité, frustum
Le LOD uniforme provoque des pertes d'informations sur les propriétés matérielles de l'objet. Avec le LOD sélectif, ces propriétés sont conservées.

Sequence d'opérations

Les premiers travaux sur les niveaux de détails sélectifs furent proposés par Hoppe [Hop97] d'un côté, Varshney et Xia [XV96] de l'autre. Ils se basent sur la méthode de stockage en mémoire des *progressive meshes* et proposent comme nous venons de le voir de nombreux critères pour pouvoir créer des niveaux de détails sélectifs.

De plus, ils introduisent la notion de *merge tree*, arbre permettant de choisir le niveau de détails affiché. Chaque nœud modélise une opération de simplification/raffinement, chaque arc représente une dépendance entre les opérations. Un niveau de détails correspond alors à la réalisation d'une séquence de nœuds en fonction de l'arbre généré.

Face Clusterisation

Garland et Shaffer sont les principaux auteurs proposant des niveaux de détails sélectifs utilisant des clusterisations. Ils utilisent dans [GWH01] l'erreur métrique quadrique définie par leurs soins dans [GH97]. Leur but est de créer une hiérarchie de surfaces pouvant être utile dans de nombreuses applications et non uniquement pour la simplification de maillages.

Pour cela, ils définissent une clusterisation de sommets. Ils réalisent une séquence de *Full-Edge Collapse* qui permet à chaque effondrement, par dualité, d'étendre un cluster. L'effondrement est réalisé sous une condition de planarité qui exclue la fusion de possibles recouvrements de la surface. De plus, ils imposent une topologie de disque à leurs clusters en ajoutant une notion de régularité des clusters.

Cet algorithme est itératif et stocke en mémoire à chaque pas la clusterisation obtenue. On obtient alors un modèle multirésolution.

Forêt de nœuds

De Floriani et Cignoni proposent un algorithme général pour l'approche multirésolution [DFMP98] qu'ils adapteront par la suite pour les maillages volumiques [CDFM+02]. Leur programme appelé *multi-tesselation* ou *MT*, est facile à utiliser car l'utilisateur ne peut spécifier que deux mesures d'erreur : une de seuil permettant de définir le degré de simplification et une de focalisation pour indiquer la zone d'attention où le maillage sera raffiné, le reste, à l'extérieur, restant grossier.

Plus en détails, l'algorithme est construit autour d'une forêt de nœuds et d'arcs qui encodent les transformations et les dépendances en indiquant quels sont les triangles à ajouter. Un *front* courant permet de savoir où l'on se situe dans la forêt et regroupe l'ensemble des triangles affichés à l'écran. Modifier le front, revient à modifier le maillage affiché.

Cette méthode propose de plus trois extracteurs de maillages qui permettent de générer des niveaux de détails. Les deux premiers sont dits *statiques* car ils repartent à chaque itération des nœuds racines et permettent d'afficher soit la totalité du maillage, soit uniquement la partie raffinée se situant dans la zone de focalisation. Le dernier est dit *dynamique* car à chaque itération, il repart du front courant pour réaliser les mises à jour. On dit alors qu'il est basé sur la *cohérence temporelle* ce qui lui permet dans la majorité des situations (quand la zone d'intérêt change peu d'une frame à l'autre) d'être plus rapide qu'un extracteur statique.

Cet algorithme, de plus, a été adapté sur une machine client/serveur [DFMMP00] afin de pouvoir transmettre les données extraites via le réseau. La communication via un réseau filaire (et encore plus via un réseau sans fil) est un goulot d'étranglement en temps. De Floriani et al. propose donc ici quelques idées afin de réduire le temps de communication afin d'obtenir un algorithme presque aussi performant que si on l'exécutait sur une seule machine.

La première idée est d'utiliser en même temps la cohérence temporelle et un algorithme de compression de données afin de minimiser le flot de données échangé entre les deux machines.

La seconde idée est de mettre en place un système de caches afin d'éviter la retransmission de données acquises quelques frames auparavant.

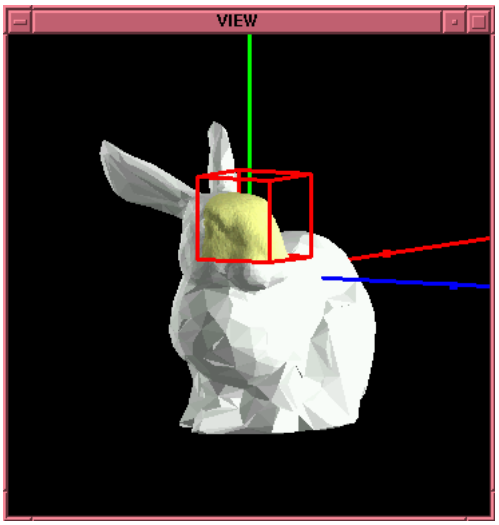


Figure 6 : Utilisation de MT avec une zone de focalisation cubique
Le maillage est grossier en dehors de la boîte (en blanc) et fin dans la boîte (en jaune). Au niveau de la frontière, on trouve des triangles qui deviennent de plus en plus grossiers en s'éloignant. Cela est dû aux relations de dépendances entre les opérations de raffinement qui forcent certains dédoublements de sommets qui ne se trouvent pas dans la zone de focalisation.

Visualisation de terrains

De nombreux autres algorithmes ont été développés se spécialisant pour la visualisation de grands terrains. En effet, par exemple, le survol d'une grande étendue par un objet volant est une application entrant dans cette problématique. Le terrain étant modélisé par des milliers de triangles, il devient nécessaire de pouvoir définir un cône de focalisation dans lequel un terrain haute définition sera affiché alors qu'aux alentours, il sera épuré avec des faces grossières.

Duchaineau and al. [DWS+97] proposent ainsi un algorithme très efficace en coût mémoire et en temps d'affichage spécialisé pour la visualisation de terrains. Ces travaux sont importants car ils seront repris par la suite pour développer des méthodes *en mémoire externe* que nous détaillerons par la suite.

Cet algorithme se base sur la méthode de raffinement *triangle bintree* et sur la notion de cohérence temporelle, c'est-à-dire prendre les résultats précédents comme point de départ d'une nouvelle recherche.

La principale astuce est d'utiliser deux piles à priorité qui permettent de prendre en compte cette cohérence temporelle. Une pile stocke les opérations d'effondrement et la seconde celles de raffinement. Elles mettent à jour ainsi à chaque itération de l'algorithme l'ensemble des arêtes pouvant être effondrées ou créées.

1.3 Traitement en mémoire externe

Les traitements en mémoire vive (*in-core*) se sont rapidement développés comme nous venons de le voir. Mais l'amélioration soudaine des outils de capture, comme les scanners, à la fin des années 90 a provoqué l'arrivée de gros maillages de données (plus de 10 millions de sommets) ne pouvant pas être visualisés par l'ensemble de ces algorithmes sur des stations de travail usuelles.

Il a donc fallu développer des techniques de visualisation et de multirésolution en mémoire externe. Nous proposons donc ci-dessous de détailler certains de ces algorithmes. L'article [SCESL02] propose un tour d'horizon détaillé des techniques de visualisation en mémoire externe.

Ces algorithmes de multirésolution reposent bien entendu sur des algorithmes de simplification. La méthode de clusterisation que nous avons détaillée précédemment (cf. section 1.1.2) est de loin la méthode la plus utilisée car elle est rapide et simple à implémenter dans ce cadre. En effet, il suffit de construire une grille régulière dans l'espace et de voir chaque voxel comme un cluster donc un sommet grossier. Pour plus de précisions sur ces méthodes, on peut aussi lire [Ber02].

Enfin, comme toutes méthodes en mémoire externe, ces algorithmes sont coûteux en temps car nécessitent l'appel fréquent des entrées/sorties de la machine. Nous verrons donc dans un premier temps, les méthodes pour limiter ce nombre d'appels, puis nous présenterons quelques algorithmes de multirésolution et de visualisation entrant dans ce domaine.

1.3.1 Limiter la complexité en temps

Les algorithmes en mémoire externe permettent de traiter de grosses masses de données car ils utilisent des fichiers situés sur le disque dur de la machine ce qui limite l'utilisation de la mémoire vive. Pourtant, ils ont un inconvénient majeur : la communication avec le disque dur se fait via des commandes d'entrée/sortie qui sont coûteuses en temps. Or, un des buts principaux de la visualisation est de pouvoir afficher interactivement les maillages. Il a fallu donc trouver des méthodes permettant de minimiser ce temps d'entrée/sortie.

Nous vous en présentons quelques-unes ci-dessous rappelées par Isenburg dans [ILGS03].

Découpage du maillage : *Mesh cutting*

Cette méthode a été la toute première utilisée pour faire du traitement en mémoire externe sur des gros maillages de données. Proposée par Hoppe pour la visualisation de terrains [Hop98], elle consiste à découper le maillage en morceaux assez petits pour pouvoir être chargés en mémoire vive. Chaque morceau est traité séparément et les arêtes de bord subissent un traitement spécial.

Cette méthode simple présente pourtant des inconvénients. Le découpage peut être très cher et la qualité du maillage obtenu peut être remise en cause car le maillage n'est pas traité comme un tout.

Traitement en blocs : *Batch processing*

Cette méthode tente de limiter l'occupation mémoire tout en maximisant de manière continue celle du processeur en lui envoyant en plusieurs passes des données. De plus, les operations sont restreintes uniquement à la quantité de données en mémoire.

Pour cela, on utilise souvent des soupes de polygones sans information de connexité ce qui augmente la taille du fichier de départ et demande un précalcul pouvant être assez lent. De plus, les résultats du fait de ce manque d'informations sur la connexité peuvent être moins corrects que ceux utilisant ces informations.

Traitement en ligne : *Online processing*

Cette méthode consiste à accéder à une certaine quantité de données via des requêtes. Afin de limiter les coûts d'accès, les données sont réarrangées en fonction des requêtes qui seront effectuées. Des caches peuvent être de plus utilisés afin d'accélérer les communications.

On utilise souvent des opérations de "mapping" pour simuler une mémoire virtuelle avec pagination. Mais ces méthodes sont restreintes à un nombre de pages maximal lié à l'architecture de la machine. Par contre, les informations de connectivité sont utilisables ce qui donne de meilleurs résultats que les deux méthodes précédentes même si le traitement en ligne est de ce fait plus coûteux en temps et en mémoire à cause de la taille des structures de données.

	Avantages	Inconvénients
Mesh Cutting	simple	traitement indépendant, perte de qualité coûteux en temps
Batch Processing	faible utilisation de la mémoire	pas de connexité fichiers d'entrée gros
Online Processing	pagination, mémoire virtuelle connexité	dépendant de l'architecture de la machine coûteux en temps et mémoire

Figure 7 : Comparaison entre les trois méthodes de traitement en mémoire externe

1.3.2 Multirésolution et Visualisation

Les algorithmes en mémoire externe de multirésolution et de visualisation se basent principalement sur les méthodes que nous avons précédemment détaillées. Garland, Shaffer, Lindstrom et Isenburg sont les principaux auteurs de ce type d'algorithmes, algorithmes que nous présentons ci-après.

Clusterisation

Nous présentons trois algorithmes qui utilisent la clusterisation pour réaliser de la multirésolution en mémoire externe.

Le premier a été conçu par Lindstrom [Lin03]. Il réalise tout d'abord une clusterisation à l'aide d'une grille régulière afin de simplifier le maillage fin initial avec une erreur métrique quadrique. Puis, il contruit une hiérarchie multirésolution en mémoire externe. Pour cela, il utilise un *octree* basé sur le tri des triangles initiaux en fonction de leur place dans la grille régulière. Les relations de dépendances entre les nœuds père et les nœuds fils sont déterminés grâce à la création d'un fichier temporaire puis à la traversée complète de l'octree. La hiérarchie permet alors d'accéder aux triangles grossiers vers les triangles fins.

L'affichage est géré par la pagination des nœuds actifs de l'octree. En effet, on utilise un *front* (ensemble des nœuds actifs à une frame donnée) pour se déplacer dans l'octree.

Le second a été présenté par Yoon et al. [YSGM04]. Il utilise une hiérarchie de clusters de maillages progressifs. Elle est calculée en mémoire externe en incluant la décomposition, la génération de la hiérarchie et la simplification des clusters. L'originalité de cet algorithme est de créer des clusters équilibrés, c'est-à-dire avec le même nombre de sommets en moyenne. La hiérarchie se veut de plus équilibrée avec un minimum de sommets partagés. Chaque cluster est ensuite simplifié d'une manière indépendante tout en veillant aux contraintes de bord (les sommets partagés) qui ajoutent des dépendances entre les clusters, afin d'éviter des *poppings*.

L'affichage est alors assuré en stockant en mémoire vive le maillage grossier ainsi qu'une partie des clusters. Deux threads sont utilisés pour dissocier la mise à jour, donc le chargement de données, et le rendu.

Le dernier a été proposé par Shaffer et Garland [SG05] et ne se restreint pas au domaine de la multirésolution puisqu'ils proposent aussi des solutions pour la détection de collision. Là aussi, ils utilisent un octree et, en plus, un tri des sommets. La fusion de cellules de l'octree permet d'obtenir des surfaces plus grossières. En entrée, ils fournissent soit une soupe de polygones, soit un maillage indexé selon les faces. Ils découpent ensuite leur algorithme en trois passes : la clusterisation des sommets, puis celle des faces et enfin une finalisation qui place correctement le sommet représentatif de chaque cellule de l'octree.

Pour le rendu, les fichiers en mémoire externe sont paginés en mémoire vive. On utilise là encore un front qui permet de mettre à jour les données, ce front ne pouvant se déplacer que d'un niveau par frame afin de garantir dans toutes les situations une interactivité au détriment de la correction de l'affichage.

Half-Edge Collapse

Cet algorithme proposé par DeCoro et Pajarola [DP02] se base sur l'opération de simplification *half-edge collapse* et une hiérarchie binaire sous forme de forêt d'arbres fusionnés. Cette hiérarchie est découpée en blocs pour ensuite être paginée et chargée efficacement.

Plus précisement, toutes les données sont stockées dans un gros fichier dont l'entête contient toutes les informations pour retrouver les blocs. Ceux-ci sont de taille variable afin d'optimiser l'emplacement mémoire. Le fichier est construit de la manière suivante. Le maillage grossier est d'abord

stocké. Puis en faisant deux passes en profondeur, on détermine les opérations de simplification en premier puis les blocs en les renumérotant. Les blocs sont ensuite écrits et enfin les nœuds racines sont déterminés.

Pour l'affichage, les blocs sont chargés s'ils contiennent des raffinements directs ou contraints (par des relations de dépendances dues à l'utilisation de *half-edge collapse*). Ils sont stockés dans une pile qui effacent les premiers entrés quand celle-ci est pleine. Là encore, un front est parcouru afin de mettre à jour le maillage.

MT

Cignoni et al. proposent dans [CGG+05] une version en mémoire externe de leur *multi tessellation*. Ils redéfinissent entièrement la conception de leur algorithme pour pouvoir l'appliquer à des gros maillages de données. Ils construisent des séquences de plus en plus fines de partitions du maillage de départ et établissent entre chaque zone d'une partition, les liens de dépendances avec sa partition mère. En composant habilement l'ensemble de ces partitions, ils peuvent construire un maillage multirésolution.

Ces partitions peuvent être définies en utilisant soit une grille régulière, soit en utilisant des triangles "graines" qui étendent, selon un rayon fixé et certaines conditions, leur zone d'occupation.

L'algorithme utilise ensuite deux threads pour pouvoir extraire et afficher séparement les partitions. De plus, deux tas, un de simplification et un de raffinement, sont conservés et mis-à-jour à chaque itération afin d'assurer une cohérence temporelle (comme vu dans la section 1.2.2).

Cette version a de plus été optimisée pour une utilisation maximale de la carte graphique.

Chapitre 2

Contributions

Nous avons précédemment rappelé les différents travaux portant sur la visualisation de gros maillages surfaciques et volumiques. Ceux-ci se sont au cours de la décennie adaptés aux nouvelles tailles des données passant d'un traitement en mémoire vive aux premiers algorithmes en mémoire externe. L'ensemble de ces travaux permettent de visualiser les maillages en faisant un compromis entre résultats visuels corrects et une interactivité d'utilisation.

Bien que la majorité des algorithmes permettent d'afficher de "beaux" et "corrects" maillages malgré de possibles simplifications, ceux-ci ne présentent pas, ou que très rarement, des résultats permettant une véritable interactivité dans l'utilisation de l'outil de visualisation. Quand celle-ci est présente, elle se fait souvent au détriment d'un résultat correct immédiat dans la zone de focalisation. En effet, celui-ci sera adapté seulement quelques frames plus tard afin de garantir l'interactivité.

Nous proposons donc ici de nous intéresser à une **visualisation interactive** dans tous les cas, pour tous types de maillages (surfaciques ou volumiques). L'aspect de l'objet en dehors de la zone d'intérêt dite de focalisation n'étant pas primordial. Par contre, en son sein, le maillage devra être affiché avec la meilleure précision, c'est-à-dire avec le maillage original, avant simplification.

Dans un premier temps, nous présentons l'algorithme que nous proposons pour répondre à cette problématique ; puis nous détaillons plus précisement chaque étape cruciale. Nous discutons enfin des possibles améliorations réalisables en détaillant ce qui a déjà été effectué.

2.1 Présentation générale

Rappelons que nous souhaitons obtenir une visualisation interactive de gros maillages surfaciques et volumiques afin de pouvoir les explorer aisément.

La taille des données d'entrée, maillages de millions de sommets, ne nous permet pas de les afficher dans leur résolution originale. Nous devons donc les simplifier afin de diminuer amplement le nombre de triangles à afficher pour accélérer le rendu de ces maillages.

Nous souhaitons aussi visualiser localement une partie précise du maillage. Cette zone d'intérêt permet l'exploration du maillage basée sur la forme approximative donnée par le maillage grossier précédemment construit. Cette zone devra être rendue le plus finement possible afin d'être la plus utilisable possible. En effet, c'est dans cette région que l'utilisateur recherche des données susceptibles de l'intéresser.

Nous avons ainsi défini deux maillages distincts : un maillage grossier global permettant de donner l'aspect général et approximatif de l'objet visualisé ; et une zone locale d'intérêt de taille variable, mais petite par rapport à l'ensemble de l'objet, permettant d'afficher tous les détails nécessaires à l'utilisateur dans cette région. Nous devons ensuite reconnecter ces deux maillages distincts afin d'en avoir plus qu'un. Cela devra fonctionner rapidement dans tous les cas, pour les maillages surfaciques comme volumiques.

On peut donc résumer l'algorithme de visualisation que nous proposons en trois étapes distinctes :

1. Construire un maillage grossier global pour esquisser l'aspect général du maillage.
2. Définir puis construire une zone locale fine d'intérêt pour afficher des informations utiles là où l'utilisateur en ressent le besoin.
3. Fusionner ces deux maillages d'une manière générale et efficace.

Notons que si l'étape 1 peut être réalisée en préprocessing, les étapes 2 et 3 doivent être interactives. Nous détaillons ces trois étapes par la suite dans le cadre de maillages surfaciques (sauf exceptions). Un raisonnement identique est tenu pour les maillages volumiques.

2.2 Construire un maillage grossier

La première étape essentielle de l'algorithme est la simplification du maillage fin initial en un maillage final dit grossier. En effet, l'affichage du maillage initial, composé de millions de sommets et donc de triangles, ne permettrait pas une application interactive à cause du goulot d'étranglement existant entre le CPU et la carte graphique. Malgré de nombreuses améliorations dans le domaine des cartes graphiques, ce goulot d'étranglement persiste et la seule solution envisageable est donc de diminuer le flot de données entre les deux entités.

Nous avons vu lors de l'état de l'art que de nombreuses techniques permettaient d'obtenir un maillage grossier. Nous utilisons ici la clusterisation pour deux raisons principales. La première est que la clusterisation est en fait une généralisation de toutes les techniques de simplification. La seconde est qu'elle est simple à implémenter autant pour les maillages surfaciques que volumiques, autant pour les méthodes en mémoire interne qu'externe.

La construction du maillage grossier à partir d'une clusterisation sera donc détaillée, mais on peut déjà exposer de manière intuitive qu'un cluster est équivalent à un sommet du maillage grossier final.

2.2.1 Clusterisation

Pour obtenir une clusterisation des maillages, nous n'avons pas procédé de la même manière avec un maillage surfacique et un maillage volumique car les données que nous possédions étaient

différentes. Nous présentons donc ici les deux approches en mémoire interne que nous avons utilisées. L'approche en mémoire externe sera développée par la suite.

METIS

Pour les maillages surfaciques, nous avons utilisé l'algorithme proposé par Karypis et Kumar [KK98]. Il fonctionne en deux temps. Dans une première étape, il crée un fichier intermédiaire à partir d'un fichier contenant des informations de connexité sur le maillage. Plus précisement, le fichier fournit en entrée est construit de telle sorte que chaque triangle connait ses trois voisins : on a ainsi des relations d'adjacence utiles pour le déroulement de l'algorithme.

Dans une seconde étape, le fichier intermédiaire permet la création de deux nouveaux fichiers, l'un contenant une clusterisation des faces et l'autre, une clusterisation des sommets. C'est ce dernier qui nous intéresse. Notons que le nombre de clusters obtenus est préalablement choisi par l'utilisateur. Ce critère est important car il va permettre de définir le degré de simplification du maillage grossier final.

Séquence d'*edge collapse*

Pour les maillages volumiques, nous reprenons des résultats obtenus par Fabien Vivodtzev au cours de sa thèse sur la simplification de maillages volumiques avec conservation de la topologie et de sous-structures incorporées. La simplification de ces maillages est obtenue grâce à une séquence d'*half edge collapse* qui est stockée dans un fichier.

Or, une telle séquence est équivalente à une clusterisation comme l'a démontré Garland dans [GWH01]. Expliquons plus précisement cette équivalence grâce à la figure 8. Cette figure présente à gauche une série d'effondrements d'arêtes : $1 \rightarrow 2$, $8 \rightarrow 2$ et $26 \rightarrow 2$. Nous obtenons donc à la fin un maillage simplifié ne contenant plus que trois sommets : le 2, le 13 et le 42. Nous pouvons voir ce résultat, si on admet qu'un sommet grossier est équivalent à un cluster du maillage fin, comme une clusterisation du maillage initial composée de trois clusters différents : un cluster identifié par le sommet 2 et contenant les sommets 1, 2, 8 et 26 ; et deux clusters contenant uniquement leur identificateur : 13 et 42.

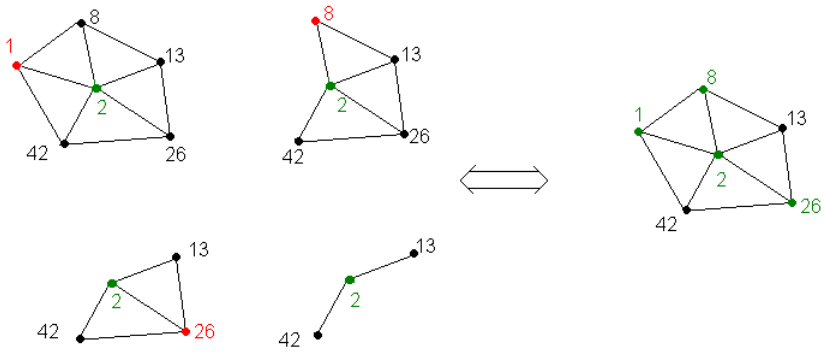


Figure 8 : D'une séquence d'edges collapse vers une clusterisation.
Ici la séquence $1 \rightarrow 2$, $8 \rightarrow 2$, $26 \rightarrow 2$ à gauche, crée sur ce simple maillage trois clusters dessinés à droite : le cluster vert identifié par le sommet 2, les deux clusters noirs différents 42 et 13.

2.2.2 Nerf

Afin d'introduire correctement la construction du maillage grossier à partir de la clusterisation, il est nécessaire d'énoncer quelques notions de topologie algébrique. Nous allons donc détailler la notion de complexe simplicial et de nerf. Pour plus de détails sur la topologie algébrique et les maillages, on peut se reporter aux cours de Edelsbrunner [Ede99].

Un *simplexe* σ de dimension n est l'enveloppe convexe d'un ensemble S de $n+1$ points affinement indépendants dans un espace affine. On appelle sommet un simplexe de dimension 0, arête un simplexe de dimension 1, triangle un simplexe de dimension 2 et tétraèdre un simplexe de dimension 3. Par convention, l'ensemble vide est dit de dimension -1.

Toute enveloppe convexe d'un sous-ensemble de points $T \subseteq S$ définit une *face* τ du simplexe σ issu de S. En particulier σ et $\emptyset$ sont des faces de σ.

Un *complexe simplicial* K est une collection finie de simplexes telle que :
(i) $\sigma \in K$ et τ face de $\sigma \Rightarrow \tau \in K$
(ii) $\sigma, v \in K \Rightarrow \sigma \cap v$ face de σ, v (peut-être vide)

On définit enfin le *nerf* $N(\mathcal{C})$ d'une collection $\mathcal{C}$ de sous-ensembles d'un ensemble X comme un complexe simplicial abstrait dont :
- les sommets sont les éléments de $\mathcal{C}$.
- les simplexes sont une sous-collection finie $\{C_1, \ldots, C_n\}$ de $\mathcal{C}$ vérifiant : $C_1 \cap C_2 \cap \ldots \cap C_n \neq \emptyset$

Illustrons cette définition en dimension 2 grâce à la figure 9. On associe à chaque ensemble un sommet. A l'intersection de deux ensembles, une arête. A l'intersection de trois ensembles, un triangle. On obtient donc ainsi un complexe simplicial de dimension 2.

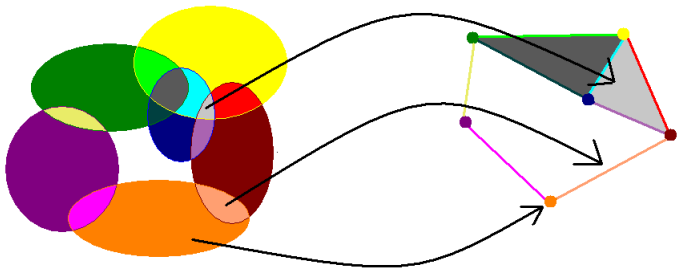


Figure 9 : Nerf associé à une collection de sous-ensembles.

En ajoutant des intersections entre quatre ensembles, on construit des tétraèdres. On peut généraliser ce résultat de la même manière aux dimensions supérieures.

2.2.3 Obtention du maillage grossier

Nous pouvons donc maintenant construire le maillage grossier.

Le maillage fin a été clusterisé selon les sommets. Notons $(C_j)_j$ l'ensemble des clusters obtenus et $(s_{i,j})_i$ les sommets fins appartenant au cluster C_j, un sommet fin n'appartenant qu'à un seul cluster.

Si on associe à chaque cluster de sommets C_j, l'ensemble T_j des triangles $(t_{ij})_i$ dont au moins un sommet s_{ij} appartient à C_j, la clusterisation induit alors un recouvrement de la triangulation. Un triangle appartiennt à T_j si tous ses sommets appartiennent à C_j. Il appartient à $T_i \cap T_j$ si au moins de ses sommets appartient à C_i et un autre à C_j, à $T_i \cap T_j \cap T_k$ si chacun de ses trois sommets appartient à un cluster différent, respectivement C_i, C_j et C_k. Notons qu'un triangle ne peut être au mieux qu'à l'intersection de trois clusters.

Les $(T_j)_j$ définissent un recouvrement du maillage fin. On peut donc constuire le nerf associé aux $(T_j)_j$. Le nerf est défini de telle sorte qu'un sommet grossier est associé à chaque cluster C_j ; une arête grossière à chaque intersection $C_i \cap C_j$, $i \neq j$, non vide ; un triangle à chaque intersection $C_i \cap C_j \cap C_k$, $i \neq j, j \neq k, i \neq k$, non vide. On associe ainsi à un maillage triangulé fin un complexe simplicial de dimension 2 abstrait grossier.

En pratique, on associe à chaque cluster C_j, un sommet grossier g_j se situant à son barycentre. Ensuite, il suffit de conserver les triangles apparteant à $C_i \cap C_j \cap C_k$ dans le maillage fin. Ces triangles sont alors agrandis en faisant glisser chaque sommet fin s_{ij} sur le sommet grossier g_j associé au cluster correspondant C_j. On obtient alors un maillage grossier sans connectivité, ce qui est suffisant pour l'application. Notons que rien ne garantit que ce maillage forme une variété sauf si les clusters sont obtenus à partir d'opérations de simplifications du type *edge collapse*, comme expliqué dans le paragraphe 2.2.1.

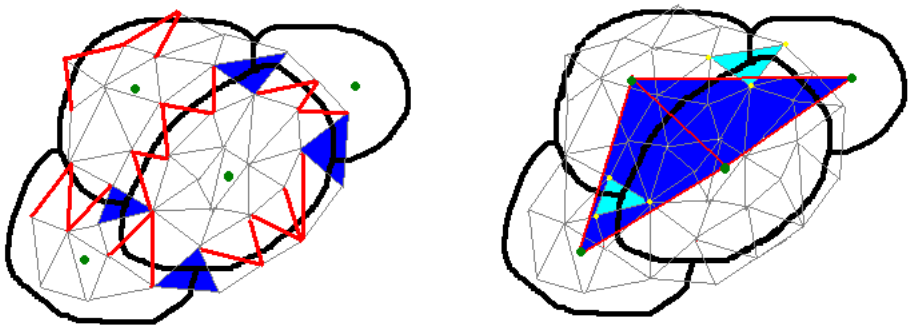


Figure 10 : Création du maillage grossier.
A gauche, on repère les intersections (arêtes en rouge, triangles en bleu) et on associe un sommet (vert) à un cluster. A droite, on construit le maillage grossier en repérant les triangles fins dans l'intersection de trois clusters (bleu ciel) et on les agrandit sur les sommets grossiers (bleu foncé).

2.3 Extraire une région fine d'intérêt

La seconde étape essentielle de l'algorithme est la définition puis la construction d'une zone locale d'intérêt affichant le maillage fin initial. Cette zone va permettre à l'utilisateur de visualiser très précisement un endroit de son choix qu'il pourra repérer grâce au maillage grossier précédemment construit.

Notons que l'algorithme rentre dans le domaine de la multirésolution. Mais contrairement aux travaux précédents, seulement deux résolutions du maillage visualisé sont utilisées : la plus fine

initiale et la plus grossière issue de la clusterisation. Il faudrait alors parler plus justement de bi-résolution.

2.3.1 Définition de la région

Tout d'abord, il faut définir la région de visualisation. Celle-ci est spécifiée grâce à une face centrale et une distance.

L'utilisateur définit ainsi son centre d'intérêt lors de la manipulation en déplaçant un pointeur à l'aide de sa souris. La forme de la région dépend de la distance choisie. Avec la distance euclidienne, on choisit l'intersection du maillage avec une sphère centrée sur la face centrale. Mais on peut tout aussi bien choisir la norme L_∞ qui revient à considérer l'intersection entre le maillage et un cube ; ou toute autre norme permettant de définir une intersection. Dans un souci d'efficacité, la distance euclidienne est utilisée et la définition d'un rayon permet de spécifier la taille de la région d'intérêt.

Enfin, grâce à l'utilisation du clavier, le rayon de la zone d'intérêt peut être agrandi ou diminué selon la volonté de l'utilisateur.

2.3.2 Mise à jour de la région

Contrairement à la construction du maillage grossier qui est réalisée auparavant en pré-traitement, celle de la région fine d'intérêt doit être faite le plus rapidement possible afin de garantir l'interactivité de l'application. Nous détaillons ci-après comment nous utilisons la notion de cohérence temporelle afin de garantir une mise à jour la plus rapide possible.

Il y a deux cas dinstincts pour la construction de la région d'intérêt : une où elle est entièrement reconstruite et une où elle est mise à jour par rapport à la précédente. Le premier cas se produit lors de l'initialisation ou lorsque le centre d'intérêt est déplacé de telle sorte qu'il se situe en dehors de l'ancienne région d'intérêt. Le second cas se produit le reste du temps, c'est-à-dire lorsque la face centrale se trouve encore dans l'ancienne région d'intérêt.

L'algorithme de construction de la région est le suivant. Nous partons dans le premier cas de la face centrale et en utilisant les relations d'adjacences sur les faces, la région est grossie jusqu'à rencontrer les faces de "bord" de la région, c'est-à-dire les premières faces qui ne satisfont plus le critère de distance.

Dans le second cas, les faces de bord constituent le point de départ. Ensuite, dans un premier temps, les faces voisines sont parcourues tant qu'elles ne sont pas dans la nouvelle région d'intérêt. Une fois le nouveau bord atteint, une nouvelle phase d'expansion est réalisée en utilisant à nouveau les relations d'adjacence sur les faces. Ainsi à partir de l'ancienne région d'intérêt, la nouvelle région est très rapidement obtenue. Ce cas utilise ainsi ce qu'on appelle la cohérence temporelle. La figure 11 illustre cette situation.

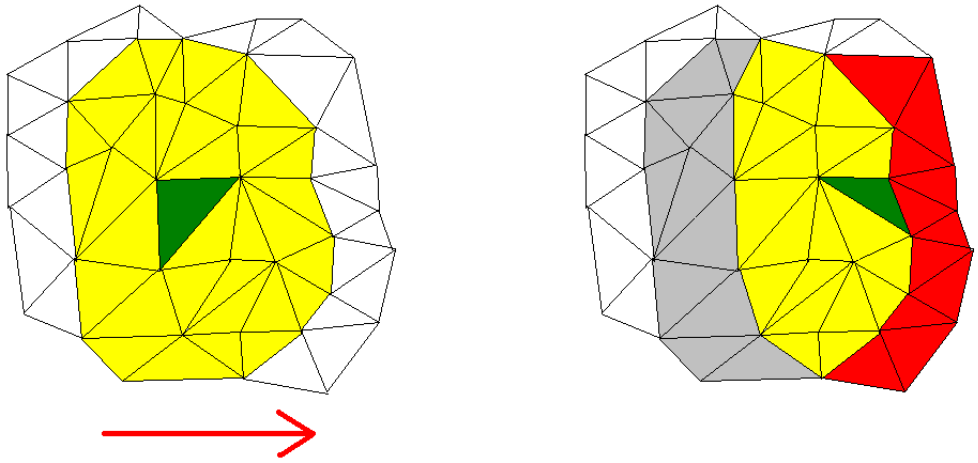
Afin de garantir l'interactivité de l'application, une liste des faces de bord ainsi qu'une table de hachage stockant l'ensemble des faces de la région d'intérêt (qui n'occupent que peu de mémoire car ne représentent qu'une très faible partie du maillage fin original) sont maintenues en permanence.

La liste permet d'avoir accès au bord de la région et donc de réaliser rapidement les phases de réduction et d'expansion de la région.

La table de hachage permet quand à elle de mettre à jour en temps constant $\mathcal{O}(1)$ pour chaque triangle la région d'intérêt autant pour l'ajout que la suppression.

L'utilisation de ces deux structures de données permet ainsi de minimiser la complexité de l'algorithme qui est de l'ordre du nombre de triangles enlevés puis ajoutés lors de la mise à jour.

Si on appelle δ le nombre de triangles enlevés puis ajoutés pour passer d'une région à une autre, alors la complexité de l'algorithme est $\mathcal{O}(\delta)$. Par exemple dans la figure 11, δ correspond à la somme du nombre des faces grises et des faces rouges donc $\delta = 12 + 11 = 23$.



avant après

Figure 11 : Construction de la nouvelle région d'intérêt en utilisant la cohérence temporelle. On déplace vers la droite le centre d'intérêt (face verte), donc la région d'intérêt initiale (faces jaunes). Lors de la mise à jour, des faces sont enlevées : les faces grises ; des faces sont conservées : les faces jaunes ; et des faces sont ajoutées : les faces rouges. La nouvelle zone d'intérêt est donc constituée des faces jaunes et des faces rouges.

2.4 Obtenir un unique maillage

Nous avons construit les deux maillages à la base de l'application. Un maillage grossier afin de pouvoir se repérer tout en diminuant sa taille pour permettre un affichage interactif et une occupation minimale en mémoire interne. Une zone locale d'intérêt utilisant le maillage fin afin d'obtenir la meilleure précision possible en son sein. Il faut maintenant pouvoir les relier afin de ne construire qu'un seul et unique maillage.

Pour cela la clusterisation liée à la construction du maillage grossier est utilisée. En effet, la région locale d'intérêt se situe sur un ou plusieurs clusters. Pour pouvoir relier proprement et surtout dans tous les cas : maillage surfacique ou volumique, maillage grossier simplifié à l'extrême (seulement quelques triangles) ; on s'intéresse aux clusters qui ont une intersection non vide avec la région d'intérêt. C'est grâce à ces clusters que le raccord entre la zone fine et la zone grossière va

pouvoir se réaliser. On nommera *intérieurs* ces clusters conservés pour l'affichage de la zone d'intérêt et *extérieurs* les clusters situés au "bord" de la région définie par les clusters intérieurs.

Le raccord se fait en deux étapes. La première étape concerne le maillage grossier. Un "trou" est créé dans lequel non seulement la région d'intérêt mais aussi l'ensemble des clusters qu'elle intersecte vont être glissés. Ce trou est réalisé en enlevant l'ensemble des triangles grossiers ayant un sommet grossier correspondant à un cluster intérieur. Un tableau de la taille du nombre de clusters est utilisé afin de marquer les clusters intérieurs ce qui permet de réaliser rapidement ce "trou".

Ensuite, la seconde étape utilise les clusters intérieurs du maillage fin pour combler le "trou". Seuls les triangles de bord de la région des clusters intérieurs sont modifiés. La figure 12 illustre la méthode du raccord. Il y a deux types de triangles de bord : les triangles ayant un ou deux sommets dans un même cluster extérieur (triangles verts sur la figure 12) et ceux ayant deux sommets dans deux clusters extérieurs différents (triangles bleus). Ces triangles vont subir une modification qui va permettre de les raccorder proprement au maillage grossier.

Les triangles appartenant au premier groupe vont être agrandis de telle sorte que les sommets fins associés au cluster extérieur soient projetés sur le sommet grossier correspondant. Les autres subissent le même sort mais chaque sommet fin est projeté sur un sommet grossier différent car ils ne sont pas associés au même cluster.

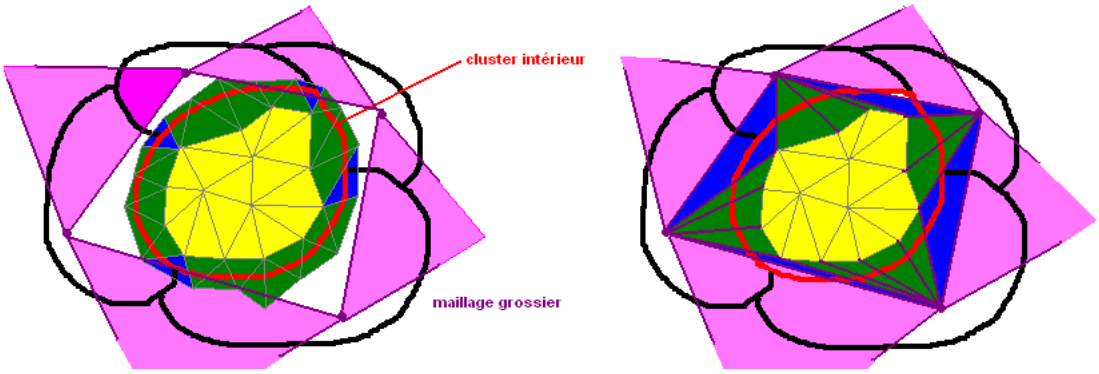


Figure 12 : Construction du raccord entre la zone d'intérêt et le maillage grossier.
A gauche, la zone d'intérêt en jaune occupe un seul cluster intérieur en rouge. Les clusters extérieurs sont en noir et le maillage grossier existant en violet. Un trou a été formé au-dessous du cluster existant. Les faces de bord sont de deux types : celles avec des sommets dans le même cluster extérieur (en vert) et celles avec deux sommets dans deux clusters extérieurs différents (en bleu). Seules ces faces sont modifiées.
A droite, on visualise le raccord. Les faces jaunes de la zone d'intérêt et les faces du maillage grossier ne sont pas modifiées. Les faces de bord sont étendues vers un seul sommet grossier pour les faces vertes (notons que les faces vertes avec deux sommets dans le même cluster sont réduites à une arête), vers deux sommets grossiers pour les faces bleues.

Ainsi, un seul et unique maillage est obtenu. Cette méthode a un intérêt majeur. Elle est, comme nous l'avons déjà précisée, très générale car elle fonctionne aussi bien pour les maillage surfaciques que volumiques même pour des maillages simplifiés à l'extrême. Un autre avantage est que cette méthode est très rapide puisqu'elle ne nécessite l'affichage que des clusters intérieurs en faisant subir une légère déformation aux triangles de bord. Ceci assure donc l'interactivité de l'application.

2.5 Visualisation distante

Une des possibles extensions de cet algorithme consiste à séparer d'un côté la visualisation du maillage grossier et les calculs de mise à jour de la région d'intérêt et de l'autre la visualisation de la région d'intérêt.

Le but est d'avoir un serveur qui met à jour rapidement la région d'intérêt et affiche sur un grand écran le maillage grossier. Le client est un pocket PC qui affiche uniquement la région locale d'intérêt. L'objectif est de réaliser une application interactive où l'utilisateur se place devant un immense écran affichant le maillage grossier et déplace le pocket PC afin de visualiser là où il le souhaite une partie précise du maillage de la taille de l'écran de son pocket PC.

Cet objectif n'a été que partiellement réalisé. La visualisation distante entre deux machines a été mise au point, mais pas l'application finale avec le grand écran et le pocket PC. L'accent a été mis en effet sur le passage en mémoire externe des algorithmes, tel que décrit dans la section 2.6.

Le principal obstacle à l'interactivité de l'application est le goulot d'étranglement dû à la communication via le réseau entre les deux machines. En effet, le flot d'émission et de réception, la taille des données mais aussi le type de transmission peuvent provoquer un retard plus ou moins important entre l'instant d'émission par le supercalculateur et l'affichage sur la machine distante.
Dans le cadre de ce stage, nous avons mis en place l'architecture permettant la communication entre deux machines et nous avons tenté de connaître qu'elles étaient les meilleures conditions pour limiter le goulot d'étranglement sans aucune modification des données ou utilisation de caches.

2.5.1 Transmission des données

Détaillons d'abord rapidement la méthode utilisée pour créer la communication entre les deux machines. Le serveur calcule à partir d'une position donnée la région locale d'intérêt fine qui va être ensuite transmise à la machine distante. La cohérence temporelle est utilisée afin de limiter le nombre de données échangées.

Plus précisement, chaque face est transmise avec un flag indiquant si elle a été ajoutée ou enlevée, son numéro de face ainsi que les coordonnées de ses trois sommets (pour les nouvelles faces) afin de permettre son affichage par la machine distante et son stockage dans une table. Le principe ensuite est le suivant. Le serveur envoie par paquets un nombre de faces. Le machine distante les reçoit aussi par paquets et les stocke dans une table de hachage qui permet d'accélérer la recherche pour la suppression. Elle n'affiche alors que les faces qu'elle possède dans cette table.

L'envoi des données et leur réception sont réalisés lorsque l'application ne reçoit aucun évènement ou ordre de rafraichissement de l'affichage. (réalisés au sein de la fonction *idle()* sous `openGL`). Aucun thread n'a été utilisé.

La cohérence temporelle est utilisée lors de l'envoi. Ainsi, seules les faces enlevées et ajoutées lors de la mise à jour de la région sont envoyées via le réseau ce qui diminue considérablement le flot de données lorsque le centre d'intérêt bouge peu (voir 2.2.2). En effet, le nombre de faces envoyé après chaque mise à jour est alors δ.

2.5.2 Temps de communication

Le but de l'application est d'être interactive. Il nous a fallu donc observer les temps de communication sur le réseau afin de choisir le bon mode de communication par sockets (TCP ou UDP) et de définir le nombre de faces à grouper par envoi et par réception. Tout cela pour obtenir une communication sans perte d'informations la plus rapide possible sans utiliser la compression de données ou des caches.

Nous avons réalisé quelques expériences afin de répondre à ces problématiques. La figure 13 indique le temps de communication entre deux machines via le réseau filaire ou sans fil à l'aide de sockets TCP (*Transmission Control Protocol*). Nous avons en effet choisi ce mode de communication afin d'éviter des pertes d'informations. Ces temps de transmissions ne permettent pour l'instant pas d'obtenir une application interactive. L'utilisation de la compression de données géoméotriques devra être mise en œuvre pour réduire les temps de communications. Ce type de compression est adapté à l'algorithme de mise à jour de la région d'intérêt.

Mais, comme nous l'avons dit précédemment, l'accent a été mis sur l'implémentation en mémoire externe des algorithmes ; comme décrit dans la section suivante.

	nombre d'envoi	# triangles envoyés par envoi	# triangles reçus par réception	nombre de réception	tps min (ms)	tps max (ms)	tps moy (ms)
réseau filaire	**1**	**1000**	**1**	**1000**	**113**	**188**	**122**
	1	**1000**	**100**	**10**	**112**	**1886**	**468**
	1	**1000**	**1000**	**1**	**113**	**115**	**114**
	1000	**1**	**100**	**10**	**42**	**1854**	**413**
	1000	**1**	**1000**	**1**	**38**	**1958**	**358**
	100	**100**	**100**	**100**	**23**	**25**	**24**
	10	**100**	**100**	**10**	**6**	**7**	**6**
	nombre d'envoi	# triangles envoyés par envoi	# triangles reçus par réception	nombre de réception	tps min (ms)	tps max (ms)	tps moy (ms)
réseau sans fil	**1**	**1000**	**1**	**1000**	**111**	**111**	**111**
	1	**1000**	**100**	**10**	**111**	**1889**	**656**
	1	**1000**	**1000**	**1**	**111**	**2347**	**1863**
	1000	**1**	**100**	**10**	**187**	**1809**	**678**
	1000	**1**	**1000**	**1**	**185**	**1794**	**871**
	100	**100**	**100**	**100**	**19890**	**21217**	**20490**
	10	**100**	**100**	**10**	**1842**	**2548**	**2120**

Figure 13 : Temps de communication entre le serveur et la machine distante (temps en ms).

2.6 Visualisation en mémoire externe pour les maillages surfaciques

Une des autres extensions réalisables est de pouvoir faire fonctionner l'application sur de très gros maillages surfaciques que l'on ne peut charger entièrement qu'en mémoire externe comme certains maillages proposés par le projet `Michelangelo` [LPC+00]. En effet, la version en mémoire interne précédemment décrite, ne peut s'utiliser que sur des maillages surfaciques ne dépassant pas les cinq millions de sommets avec une RAM de 2 Go.

Afin de pouvoir traiter ces gros maillages, nous devons mettre en place un traitement en mémoire externe qui va permettre de créer les fichiers nécessaires au bon fonctionnement de l'application. Ils sont au nombre de trois : un fichier contenant la clusterisation - c'est-à-dire associant à chaque sommet fin son sommet grossier correspondant ; un fichier contenant le maillage grossier et un fichier contenant le maillage fin avec relations d'adjacence. En effet, le maillage fin original ne possède pas de telles relations qui sont capitales pour la mise à jour de la région d'intérêt.

Une fois ces fichiers créés, l'application sera opérationnelle en paginant certains fichiers ce qui limitera l'occupation en mémoire vive. Cette pagination sera faite en tenant compte des clusters nécessaires pour le bon déroulement de l'algorithme.

2.6.1 Création des fichiers

La taille des fichiers que nous traitons est trop conséquente pour permettre un traitement en mémoire interne des données. Nous détaillons ici les différentes étapes réalisées en mémoire externe afin d'obtenir les fichiers nécessaires au bon fonctionnement de l'application.

La première étape construit des fichiers intermédiaires encodant une soupe de polygones, fichiers nécessaires pour pouvoir réaliser une clusterisation en mémoire externe. La seconde étape crée les fichiers contenant la clusterisation et le maillage grossier correspondant. La dernière étape crée un fichier contenant le maillage fin avec relations d'adjacence.

Rappelons que dans toutes ces étapes, l'accent est mis sur le traitement en mémoire externe et non sur la complexité en temps, complexité qui n'est pas primordiale puisque tout cela est fait en prétraitement.

Fichier intermédiaire : soupe de polygones

La première étape consiste à créer une soupe de polygones afin de pouvoir réaliser une clusterisation en mémoire externe par la suite.

Dans une soupe de polygones, tous les polygones sont encodés de manière à être affichés sans déréférencement. Ainsi pour chaque polygone, les coordonnées de ses sommets sont directement connues. Ici, les polygones sont des triangles. Pour ne pas perdre le lien unissant les sommets et les triangles, la *soupe* est dans notre cas encodée de la sorte. Un fichier contient la liste des sommets avec leurs coordonnées respectives. Plusieurs fichiers contiennent la liste des faces, chaque face se

référençant à ses trois sommets dont les coordonnées sont aussi précisées.

La décomposition en plusieurs fichiers de la soupe de polygones a été imposée pour pouvoir réaliser un traitement en mémoire externe. En effet, l'architecture de la machine limite la taille de l'adressage et par conséquent la taille d'un fichier ouvert en lecture ou écriture. Si un fichier est trop gros, cela peut provoquer des erreurs d'écriture et de lecture dues à un débordement de l'entier encodant l'adresse du pointeur. Le fichier obtenu dans de telles conditions serait alors faux, malgré un algorithme correct. Ce découpage en plusieurs petits fichiers est valable pour l'ensemble des fichiers crées au cours du traitement en mémoire externe pour la raison que nous venons d'évoquer.

La soupe de triangles est d'un intérêt crucial. En effet, pour le maillage grossier, il suffit de ne conserver que les triangles ayant trois sommets dans trois clusters différents. Afin d'accélerer sa construction pour éviter de longues requêtes de recherche dans le fichier initial, il est ainsi préférable que chaque triangle possède déjà l'information de sa localisation. Une soupe de triangles est donc la meilleure façon pour procéder. Elle permet ainsi de construire simultanément la clusterisation et le maillage grossier en parcourant sa liste des faces.

Clusterisation et Maillage grossier

Pour la construction de la clusterisation et du maillage grossier, une grille régulière virtuelle est utilisée. Pour constuire cette grille, la boîte englobante du maillage est déterminée. Celle-ci est calculée lors d'un parcours des sommets du maillage en récupérant les positions extrêmales selon chaque axe. Puis, elle est découpée en un nombre déterminé par l'utilisateur de cellules. Le découpage a été choisi régulier selon les axes, mais il pourrait être différent si l'on souhaitait privilégier une certaine direction pour certaines raisons. Chaque cellule définit alors un potentiel cluster de sommets fins.

La clusterisation est déterminée en parcourant la liste des faces de la soupe de polygones. Pour chaque nouveau sommet fin lu, la celulle de la grille régulière à laquelle il appartient, est recherchée. Chaque cellule non vide se voit associée alors un sommet grossier, ce qui permet de faire la correspondance entre sommet fin et sommet grossier. Une table de hachage est ainsi construite, stockant uniquement les cellules non vides. Si la cellule, donc le sommet grossier, existe dans la table de hachage, on associe au sommet fin l'indice existant. Sinon un nouvel indice de sommet grossier est créé puis ajouté à la table de hachage. Cette table de hachage stocke d'autres informations comme le placement des sommets grossiers, qui seront utiles pour construire le maillage grossier. Cela sera expliqué plus en détails ci-dessous. La clusterisation est enfin écrite en lisant le fichier de sommets fins de la soupe de polygones. Chaque sommet fin lu est alors associé à son sommet grossier grâce à la table de hachage. Un fichier est ainsi construit associant chaque sommet fin dans l'ordre lexicographique à un indice indiquant le sommet grossier, donc le cluster, correspondant.

Le maillage grossier est construit en même temps lors du parcours de la liste des faces de la soupe de polygones. Seuls les triangles ayant trois sommets dans trois cellules différentes sont conservés. Afin d'avoir un maillage visuellement "joli", le barycentre des sommets fins d'un même cluster est calculé et est utilisé comme coordonnées du sommet grossier. Ce barycentre est stocké au sein de la table de hachage décrite précédemment. Cela évite un maillage de type *légo* obtenu simplement en prenant le centre de la cellule déterminant le cluster. En effet, ce choix conserve les propriétés de la grille choisie, ce qui donne un aspect "en brique" au maillage obtenu.

D'autres méthodes pourront être utilisées afin de construire des maillages encore plus corrects topologiquement et visuellement. L'une d'entre elles sera évoquée dans le chapitre perspectives.

Maillage fin avec relations d'adjacence

Déterminer les relations d'adjacence dans un maillage nécessite d'associer à chaque arête ces deux faces correspondantes, puis en les parcourant d'associer à chaque face ses trois faces voisines. Cela implique une vision globale du maillage qui est impossible avec une méthode en mémoire externe, puisque le maillage fin ne peut être chargé entièrement en mémoire interne. Pour résoudre ce problème, les relations d'adjacence sont trouvées localement, par cluster. Les arêtes appartenant au bord des clusters sont traitées de manière globale afin de garantir les raccords entre clusters.

La création du maillage fin avec relations d'adjacence est réalisée à partir de la soupe de polygones et du fichier de clusterisation. Afin de permettre une pagination lors de la visualisation, plusieurs fichiers en sortie sont créés, leur nombre étant défini par l'utilisateur. Ces fichiers rassemblent d'un côté les sommets et de l'autre les faces. Les sommets et les faces sont regroupés au sein de ces fichiers par cluster, grâce au fichier de clusterisation. Un en-tête spécialisé est inscrit dans le premier fichier de sommets indiquant entre autres la taille des clusters de faces et de sommets ainsi que le découpage des fichiers de faces. Les fichiers de faces sont en effet découpés de telle sorte qu'aucun cluster ne se retrouve à cheval sur deux fichiers ce qui imposerait une pagination simultanée de deux fichiers pour un seul cluster. Pour l'instant, une indexation absolue est utilisée pour les faces comme pour les sommets.

Afin de faciliter l'affichage, la construction du lien et d'éviter des chargements intempestifs de fichiers de faces, les faces de bord de chaque cluster sont recopiées dans tout cluster contenant au moins un de leurs sommets. Cela revient à stocker les clusters de triangles définis en 2.2.3. Cela implique une redondance d'informations utile car la face, bien qu'identique spatialement, n'aura pas les mêmes faces adjacentes d'un cluster à un autre. Cela sert donc à éviter de visiter un cluster voisin pour une face de bord. La figure 14 illustre la configuration obtenue.

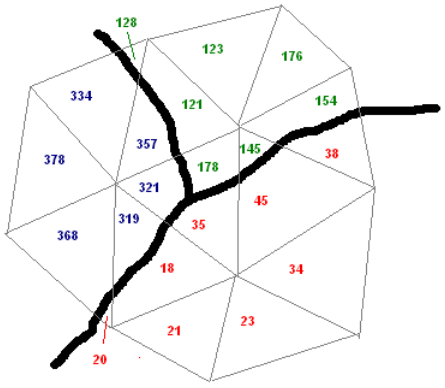


Figure 14 : Regroupement des faces par cluster.
Les faces de bord sont recopiées dans chaque cluster associé à leur sommet. Mais les relations d'adjacence sont différentes en fonction du cluster. Par exemple la face 45 a comme faces adjacentes 35, 38 et 34. Cette même face a dans le cluster voisin le numéro 145 et a pour faces adjacentes 178, 154 et 34.

Les fichiers encodant le maillage fin avec relations d'adjacence sont obtenus en appliquant quatres phases distinctes, les deux premières permettant d'initialiser les fichiers en fonction de la clusterisation et de créer un nouveau fichier de clusterisation résultant de la nouvelle indexation, les deux dernières déterminant les relations d'adjacence.

- La première phase détermine la taille des clusters de faces et de sommets. Elle écrit de plus les fichiers contenant les sommets dans l'ordre de la clusterisation à partir de la soupe de polygones et le nouveau fichier de clusterisation qui prend en compte la nouvelle indexation des sommets. L'entête est aussi inscrit lors de cette phase.
- La deuxième phase initialise les fichiers concernant les faces. Elles sont écrites en fonction de la clusterisation à partir de la soupe de polygones. Une passe supplémentaire permet de prendre en compte la nouvelle indexation des sommets par rapport à la clusterisation.
  A la fin de ces deux passes, les fichiers tiennent en compte tous de l'ordonnancement en fonction de la clusterisation.
- La troisième phase détermine la connectivité au sein des clusters en stockant les arêtes contenues dans chaque cluster de faces. Les arêtes qui ne possèdent pas deux faces associées en fin de parcours sont stockées dans une liste globale afin de pouvoir les traiter ultérieurement. Les autres arêtes permettent de mettre à jour les relations d'adjacences des faces concernées.
- La quatrième et dernière phase détermine la connectivité des arêtes stockées dans la liste globale en parcourant une nouvelle fois l'ensemble des faces du maillage fin. Chaque face du maillage connait alors ses trois faces voisines.

La création de ces fichiers va permettre à l'algorithme d'extraction de fonctionner sans apporter de grandes modifications à la version en mémoire interne.

2.6.2 Application en mémoire externe

L'application en mémoire externe est presque identique à celle en mémoire interne. Elle diffère sur deux points détaillés ci-dessous.

Pour que l'application fonctionne, le maillage grossier est chargé en mémoire interne et le fichier de clusterisation est entièrement paginé. Par contre, la pagination des fichiers encodant le maillage fin dépend de la taille de ceux-ci. Pour des maillages de moins de 10 millions de sommets, la pagination de l'ensemble des fichiers est possible. Au delà, celle-ci est impossible. Mais les fichiers avec relations d'adjacence ont été construits de façon à permettre le chargement de certains fichiers uniquement.

L'application pour résoudre ce problème fonctionne donc de la manière suivante. L'algorithme de construction de la région locale d'intérêt gère la pagination des fichiers de faces fines nécessaires à son bon déroulement en repérant quels clusters, donc quels fichiers, doivent être chargés en mémoire virtuelle. On limite ainsi le nombre des fichiers de faces paginés à un instant. Un chargement identique a aussi été implémenté pour les sommets fins utiles, afin de pouvoir faire fonctionner l'application sur le plus gros maillage testé : `David 1mm` - 28184526 sommets et 56230343 faces.

L'algorithme d'expansion et de réduction de la région locale d'intérêt est légèrement modifié pour prendre en compte les dédoublements (voire triplements) des triangles de bord des clusters. Lors de la réduction, il faut en effet veiller à enlever tous les représentants d'un même triangle dans tous les clusters. A part quelques conditions supplémentaires que cette particularité impose, le principe, le déroulement, et la complexité de l'algorithme restent inchangés.

Chapitre 3

Résultats

Nous présentons ici les résultats issus de nos algorithmes permettant une exploration interactive de gros maillages surfaciques et volumiques. Nous détaillons surtout les versions en mémoire interne et externe car celle utilisant le réseau n'est pas satisfaisante pour l'instant.

Nous rappelons que le but de ce stage est l'exploration interactive. Nous insistons donc lourdement sur ce point crucial. C'est d'ailleurs l'une des raisons qui écarte la version fonctionnant sur deux machines distantes car pour l'instant les résultats sont loin d'être interactifs.

Les résultats ont été obtenus sur un ordinateur de fréquence 2.8 GHz avec une RAM de 2 Go. L'application utilise trois types de données :

- un fichier contenant le maillage fin encodant les sommets, les triangles ainsi que les relations d'adjacence entre les faces pour les maillages surfaciques. Pour les maillages volumiques, le fichier contient en plus d'autres propriétés locales comme les matériaux, les couleurs, etc... Cela impose deux versions de chargements pour l'application malgré la généralité du raisonnement ;
- un fichier contenant le maillage grossier encodant les sommets et les faces ;
- un fichier de clusterisation qui associe à chaque sommet fin son sommet grossier correspondant dans le maillage grossier.

Notons enfin que les maillages volumiques sont issus de la thèse de Fabien Vivodtzev sur la simplification de maillages volumiques avec conservation de la topologie. Nous le remercions donc pour ses précieuses données.

3.1 En mémoire interne

Nous présentons tout d'abord les résultats obtenus en mémoire interne. Des maillages de données surfaciques et volumiques assez conséquents ont pu être traités avant de passer à la version *out-of-core*. Ainsi, la statue Thaï de Stanford possédant près de 5 millions de sommets et 10 millions de faces est visualisable au sein de cette application.

La première étape de l'algorithme est d'obtenir un maillage grossier issu de la clusterisation. On peut donc voir ci-après les résultats obtenus pour les maillages surfaciques grâce à l'utilisation de `metis` et pour les maillages volumiques grâce à une séquence d'*half edge collapse*. On notera que le

maillage volumique simplifié respecte des critères de simplification (ceux détaillés dans la thèse de Fabien Vivodtzev) comme celui de la conservation des matériaux (une couleur = un matériau dans le maillage grossier). Cela est possible grâce à l'utilisation de cette séquence de simplifications qui permet de localiser explicitement quels sont les sommets initiaux conservés dans le maillage grossier.

Rappelons enfin que le maillage grossier sert uniquement à donner une idée sur la forme de l'objet, d'une manière suffisante pour que l'utilisateur puisse se localiser facilement.

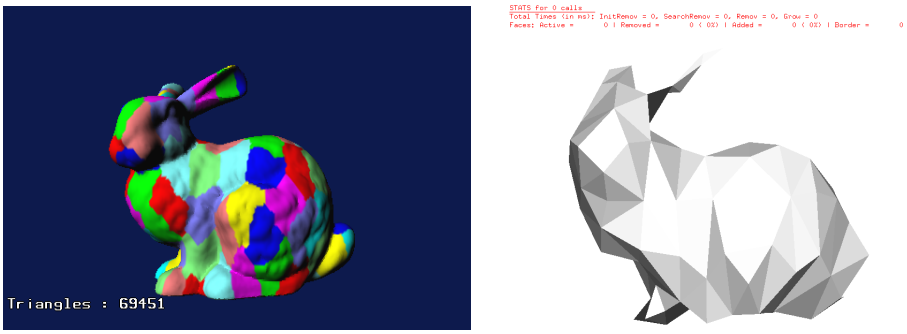


Figure 15.a : Clusterisation et maillage résultant pour un objet surfacique.

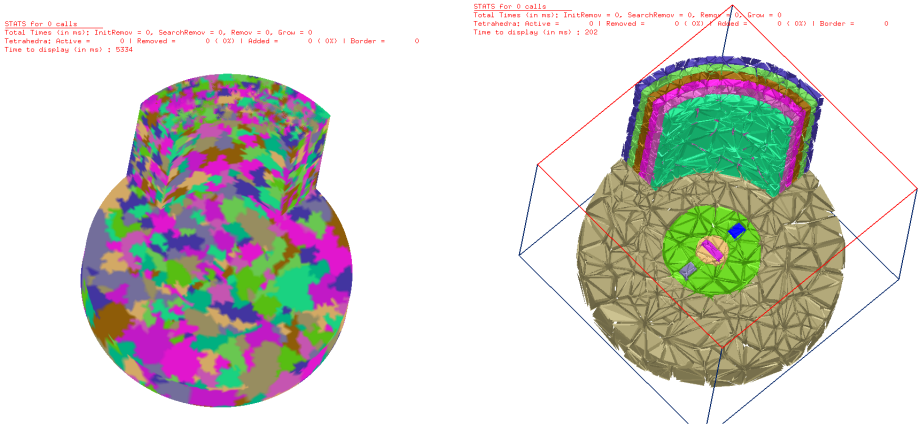


Figure 15.b : Clusterisation et maillage résultant pour un objet volumique.
Basé sur la thèse de Fabien Vivodtzev.

Les temps de calcul pour déterminer la clusterisation puis constuire ces maillages grossiers sont extrêmement rapides pour les maillages surfaciques puisque l'algorithme utilisé a été optimisé dans cet intérêt. Ils sont légèrement plus lents pour les maillages volumiques mais cela importe peu puisque cette phase est réalisée préalablement.

La première étape de l'application étant réalisée, la région locale d'intérêt doit être déterminée en temps-réel durant l'utilisation de l'application. Les figures suivantes (Figure 16.a et Figure 16.b) illustrent le principe de l'application. L'utilisateur indique son centre d'intérêt à l'aide du pointeur bleu qu'il peut déplacer aisément.

Pour la version surfacique de l'application (Figure 16.a), la région locale d'intérêt est réhaussée par une couleur jaune alors que le reste du maillage - raccord et maillage grossier - est laissé en blanc. Le but est de faciliter le travail de l'utilisateur en lui permettant de repérer rapidement la forme et la localisation de la région d'intérêt.

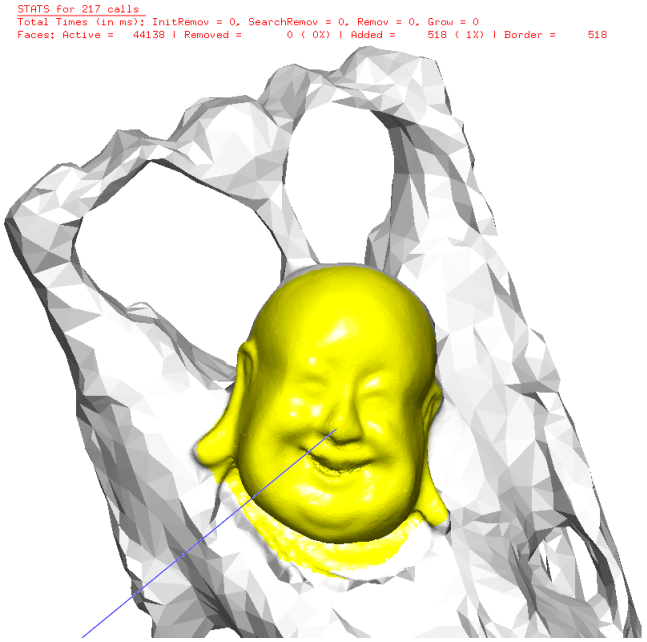


Figure 16.a : Exploration dans un maillage surfacique.

Pour la version volumique, l'application tente de rester dans l'esprit de ce que proposait Fabien Vivodtzev au cours de sa thèse. Dans la figure 16.b, chaque matériau est représenté par une couleur différente. Le maillage grossier, le raccord et la région d'intérêt reprennent ces couleurs car ce sont les informations clés de l'objet. L'affichage est ici représenté avec le raccord pour illustrer la généralité de l'approche, mais dans un souci de compréhension et de localisation, celui-ci peut ne pas être généré.

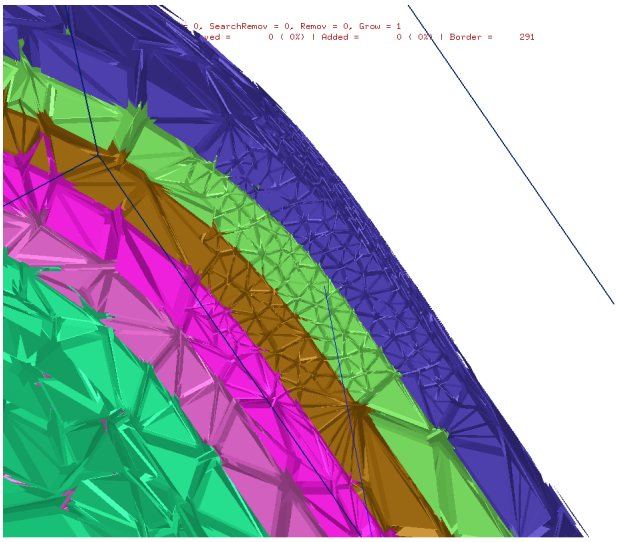


Figure 16.b : Exploration dans un maillage volumique.

L'interactivité de l'application est capitale. Elle fut testée sur de nombreux maillages surfaciques et volumiques avec une taille de la région d'intérêt pouvant varier selon le désir de l'utilisateur. Quelques résultats sont présentés dans le tableau de la figure 17.

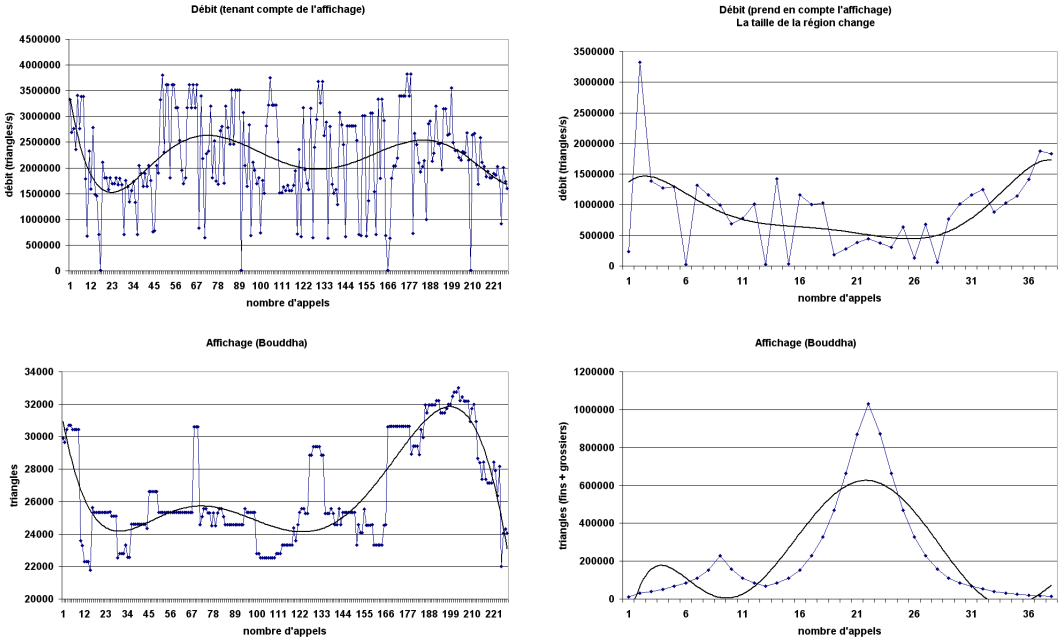


Figure 17.a : Courbes indiquant le nombre de triangles affichés en une seconde pour des maillages surfaciques. L'affichage est pris en compte dans les calculs.

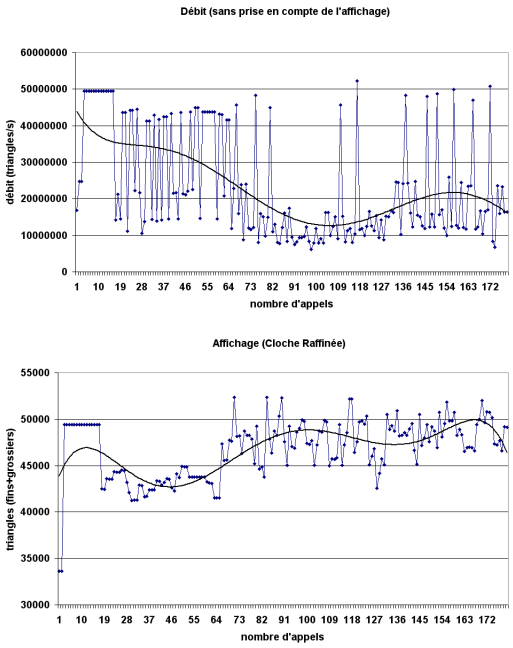


Figure 17.b : Courbes indiquant le nombre de triangles affichés en une seconde pour des maillages volumiques. L'affichage n'est pas pris en compte dans les calculs.

Pour l'exploration au sein des maillages surfaciques, le taux de rafraîchissement est en moyenne au-delà des 30 frames par seconde ce qui assure l'interactivité de l'application. Le temps de rendu a été pris en compte afin de donner une "vraie" moyenne car les temps de mise à jour de la région

fine sont inférieurs à la milliseconde dans la majorité des cas. Le débit moyen est donc de l'ordre de 2.000.000 triangles à la seconde pour un affichage moyen de 27.000 triangles grossiers et fins.

La colonne de droite de la figure 17.a indique que le débit diminue si la taille de la région d'intérêt ne fait qu'augmenter. En effet, le nombre de triangles fins ajoutés devient de plus en plus important à chaque étape. Donc les temps de calculs sont légèrement plus élévés - augmentation du δ. Le phénomène inverse se produit si on ne fait que diminuer la taille de la région.

Pour les maillages volumiques (figure 17.b), un résultat ne prenant pas en compte le rendu est fourni pour illustrer la rapiditié de l'algorithme de mise à jour de la région d'intérêt et du raccord. Ici, en effet, le temps de rendu est non négligeable du fait de maillages grossiers conséquents. Or l'étape d'affichage n'étant pas optimisée pour l'instant, elle fausse les résultats d'interactivité de l'algorithme.

On peut donc remarquer ici le véritable débit assuré par l'algorithme qui en moyenne est de 20.000.000 tétraèdres à la seconde. Ce débit assure l'interactivité de l'application.

3.2 En mémoire externe

Rappelons que l'application en mémoire externe n'a été développée que pour des maillages surfaciques. Un échantillon de maillages significatifs est utilisé pour illustrer les résultats. La majorité de ces maillages est issue de la bibliothèque `STANFORD` et certains proviennent même du projet `Michelangelo`. La figure 18 en présente les caractéristiques.

	Nombre de sommets fins	Nombre de faces fines
Bunny	35 947	69 451
Dragon	437 645	871 414
Youthful	1 728 305	3 411 563
Dragon oriental	3 609 600	7 218 906
Statuette Thaï	4 999 996	10 000 000
David 2mm	6 924 951	13 797 791
Lucy	14 027 872	28 055 742
David 1mm	28 184 256	56 230 343

Figure 18 : caractéristiques des principaux maillages surfaciques utilisés

Le temps utilisé pour la construction des fichiers représentant la soupe de polygones et le maillage fin avec relations d'adjacence n'a pas été optimisé car ces fichiers sont obtenus en précalcul. De plus, les différentes passes, lectures et écritures au sein des fichiers sont assez lentes du fait de fichiers ouverts en lecture/écriture qui imposent un positionnement récurrent du pointeur dans le fichier pour lire ou écrire au bon endroit.

Par contre, l'utilisation de la mémoire vive est assez limitée comme souhaitée. Par exemple, la place mémoire utilisée pour la transformation de la soupe de polygones vers le maillage fin avec relations d'adjacence est de l'ordre de 30 Mo au maximum pour le maillage `David 2mm` - 6.924.951 sommets et 13.797.791 faces. D'autres exemples sont donnés dans la figure 19.a.

La construction de la clusterisation et du maillage grossier associé est assez rapide car la forme des fichiers contenant la soupe de polygones a été étudiée dans ce sens. La construction de ces deux

fichiers se réalise ainsi en quelques minutes pour les plus gros fichiers. La figure 19.b donne plus de précisions sur ce point.

	Mémoire vive maximale utilisée	temps de calcul (min's")
Bunny	3 Mo	13"
Dragon	11 Mo	4'03"
Youthful	18 Mo	13'08"
David 2mm	30 Mo	47'47"
Lucy	42 Mo	120'

Figure 19.a : temps et mémoire utilisée pour la construction du maillage fin.

	Mémoire vive maximale utilisée	temps de calcul (min's")
Bunny	200 Ko	1"
Dragon	2.8 Mo	5"
Youthful	2.5 Mo	21"
Dragon oriental	2.2 Mo	40"
Statuette Thaï	2.5 Mo	57"
David 2mm	2.5 Mo	1'20"
Lucy	2.5 Mo	2'40"
David 1mm	2.5 Mo	5'20"

Figure 19.b : temps et mémoire utilisée pour la construction du maillage grossier et de la clusterisation.

Contrairement à l'application en mémoire vive qui demande un temps de chargement des données assez conséquent lors de l'initialisation, ici l'application à un temps d'initialisation quasi instantané. Les résultats visuels sont les mêmes que ceux obtenus dans la version en mémoire interne (cf figure 21). Par contre, l'occupation de la mémoire interne est beaucoup moins importante dès que le maillage contient plus de 100.000 sommets. La figure 20 illustre une telle constation en proposant sur quelques maillages représentatifs, un comparatif entre les deux applications de l'utilisation de la mémoire vive.

	En mémoire interne	En mémoire externe
Bunny	45 Mo	68 Mo
Dragon	182 Mo	72 Mo
Youthful	546 Mo	73 Mo
Dragon oriental	857 Mo	78 Mo
Statuette Thaï	1,2 Go	83 Mo
David 2mm	impossible	85 Mo
Lucy	impossible	130 Mo
David 1mm	impossible	200 Mo

Figure 20 : comparatif de l'utilisation de la mémoire vive entre les deux méthodes proposées

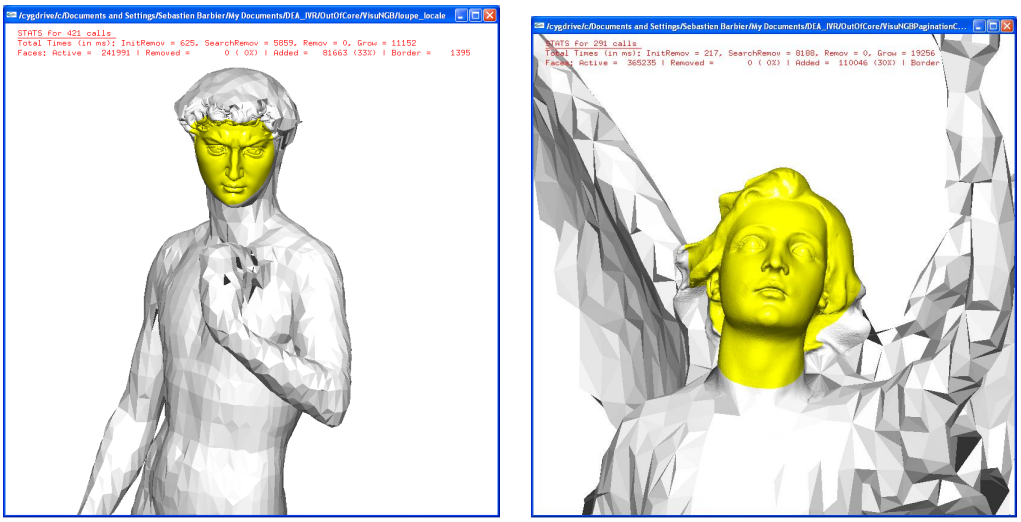


Figure 21 : Exploration en mémoire externe avec pagination de maillages surfaciques issu du projet `Michelangelo`

La figure 22 illustre enfin l'interactivité de l'application. Le débit de triangles à la seconde a été calculé sans prise en compte du temps de rendu qui pour les gros maillages de données surfaciques serait trop conséquent pour permettre de révéler la véritable complexité de l'algorithme. En effet, comme pour la version en mémoire interne, le rendu n'est pour l'instant pas optimisé. De plus, les résultats fournis sont calculés avec une version offrant une pagination complète des données afin de pouvoir les comparer de manière exacte avec ceux obtenus pour la version en mémoire interne.

Le débit de triangles est de l'ordre de 5.000.000 de triangles à la seconde pour un affichage de 250.000 triangles ce qui est un débit supérieur à celui obtenu avec affichage pour la version en mémoire interne avec le même nombre de triangles - de l'ordre de 700.000. Si l'affichage est pris en compte, un résultat légèrement inférieur à celui en mémoire interne est obtenu.

On peut là aussi remarquer la diminution du débit avec l'augmentation progressive de la taille de la région d'intérêt - c'est-à-dire que seuls des triangles sont ajoutés, et de plus en plus ; et l'effet inverse lors d'une diminution progressive.

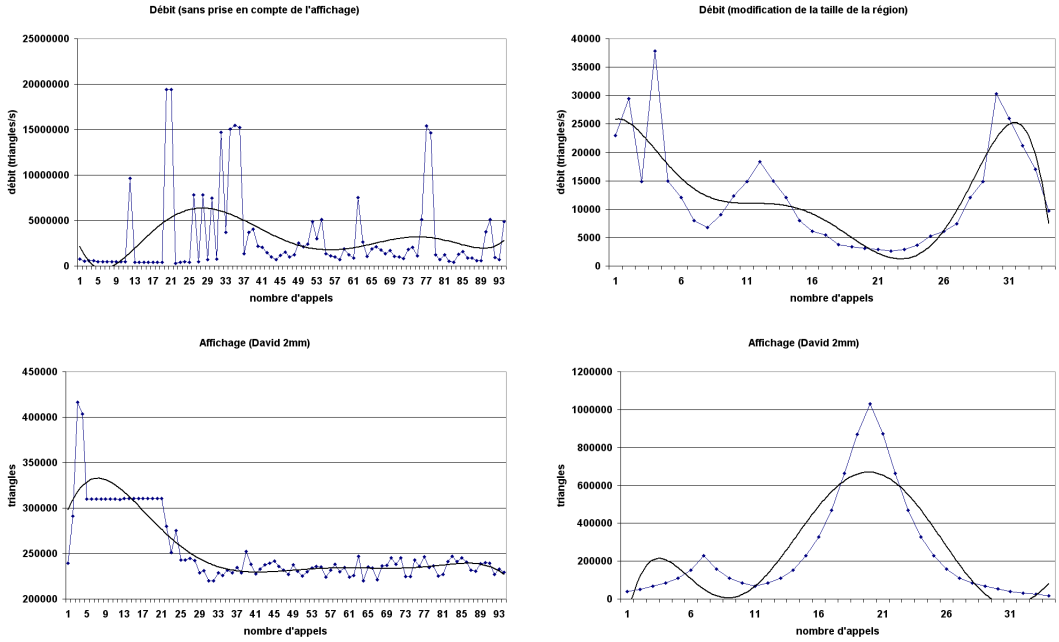


Figure 22 : Courbes indiquant le nombre de triangles affichés en une seconde pour des maillages surfaciques. L'affichage n'est pas pris en compte dans les calculs. La pagination est complète dès le départ

Chapitre 4

Perspectives

Ces travaux ne sont qu'un commencement. Nous présentons ci-après certaines améliorations et perspectives qui pourront être développées.

Une des premières améliorations possibles serait de construire un maillage grossier plus "proche" du maillage fin initial. La solution choisie pour l'instant -calcul barycentrique- crée des maillages conservant par endroits cet aspect "légo". Une méthode utilisant une notion de distance à un plan et une approximation par la méthode des moindres carrés permettrait d'obtenir des résultats sans aucun doute meilleurs.

Un effort pourra aussi être fait dans l'optique d'optimiser la complexité de l'affichage afin d'accélérer les temps de rendu pour les maillages conséquents, pour renforcer l'interactivité de l'algorithme. Des *vertex arrays* ou encore des *vertex buffer objects* pourraient être utilisés dans ce sens.

4.1 Améliorer le traitement en mémoire externe

Le traitement en mémoire externe s'est accru dans le domaine de la visualisation au cours de ces dernières années. Il reste donc un axe privilégié qui doit être amélioré par rapport aux résultats obtenus.

La pagination a été utilisée mais cette méthode a une certaine limitation car au-delà d'une certaine taille des fichiers, elle ne peut être réalisée de manière globale. Un découpage adapté des fichiers permet de ne charger qu'un certain nombre de fichiers et peut pallier à cet inconvénient. Mais il reste à déterminer réellement si cela ralentit l'algorithme d'extraction et de visualisation. D'autres solutions pourraient être alors envisagées.

Une meilleure interface permettant le déplacement du centre de la région d'intérêt pourra être aussi développée en utilisant un octree sur le maillage grossier pour se localiser plus rapidement. En effet, la solution actuelle utilisant les relations d'adjacence n'est pas idéale dans toutes les configuations, ce qui ne permet pas d'embrasser pleinement la rapidité de l'algorithme dans certains cas.

Des solutions pourront être apportées pour améliorer encore l'interactivité de l'application en

optimisant les algorithmes de recherche et de visualisation.

Enfin, les algorithmes déterminant les fichiers nécessaires à l'application pourraient être améliorés afin de diminuer les temps de calculs.

4.2 Développer la visualisation pour les maillages volumiques et hexaédriques

Les maillages volumiques ont été traités en mémoire interne de manière efficace. Un traitement en mémoire externe devra donc être développé, par généralisation du traitement pour les maillages surfaciques.

Une autre piste d'amélioration serait d'utiliser des outils de visualisation tels que les isosurfaces ou les champs de valeur et de les incorporer dans le logiciel de visualisation. Ceci sera fait dans un souci d'améliorer la compréhension et d'aider l'utilisateur dans son exploration de l'objet volumique.

La généralisation de l'algorithme a d'autres types de maillages, comme les maillages hexaédriques, pourra aussi être envisagée en mémoire interne comme en mémoire externe.

4.3 Visualisation distante

La communication entre deux machines distante doit être améliorée afin de créer une application entre un supercalculateur et un PDA qui ferait office de "loupe" en affichant des détails là où l'utilisateur le placerait devant un écran affichant le maillage grossier.

Le temps de communication entre les deux machines est à diminuer afin de permettre une véritable interactivité. L'emploi de caches et d'une compression de l'information pourra être utilisé dans ce sens.

Conclusion

Nous avons développé au cours de ce stage de M2R un algorithme permettant l'exploration interactive de gros maillages surfaciques et volumiques. Cet algorithme tente d'améliorer les résultats précédents non pas sur le domaine de la visualisation mais sur celui de l'interactivité. Dans la majorité, celle-ci n'était que peu garantie. Si celle-ci était assurée alors les résultats visuels n'étaient pas corrects dans l'instant. De plus, les maillages volumiques sont délaissés par le fait de difficiles généralisations des méthodes surfaciques.

Notre algorithme repose sur l'utilisation de deux maillages seulement : un fin précis et un grossier. Le grossier permet à l'utilisateur de se localiser et de spécifier où il souhaite voir une région locale fine dite d'intérêt. Au sein de cette région le maillage fin est affiché, en dehors le maillage grossier.

Le maillage grossier est construit en utilisant une méthode de simplification efficace : la clusterisation qui consiste à rassembler des sommets en ensembles et à associer ensuite un sommet grossier à chaque ensemble.

La construction de la région d'intérêt repose sur la cohérence temporelle et sur des structures de données efficaces afin de garantir une complexité en temps de l'ordre du nombre de triangles enlevés puis ajoutés.

Le raccord est assuré entre les deux maillages à résolution différente en utilisant la clusterisation précédemment définie.

Des applications en mémoire interne pour les maillages surfaciques et volumiques puis en mémoire externe uniquement pour les maillages surfaciques ont été développées. Elles nécessitent trois types de données : un maillage fin avec relations d'adjacence, un maillage grossier et une clusterisation associant chaque sommet fin à son sommet grossier. Ces trois fichiers sont créés en mémoire externe en utilisant peu de ressources dans le cas de la seconde application.

L'algorithme en mémoire interne assure une interactivité certaine avec un nombre de frames par seconde de l'ordre de 30. Celui en mémoire externe utilise une pagination des fichiers qui peut être limitée, et bien qu'offrant une interactivité quasiment identique lors d'une pagination complète, demande encore quelques améliorations.

Bibliographie

[Ber02] Pavel Berkin. Survey of clustering data mining techniques. *Technical Report, Accrue Software*, 2002.

[CDFM+02] P. Cignoni, L. De Flioriani, P. Magillo, E. Puppo, and R. Scopigno. Volume visualisation of large tetrahedral meshes on low cost platforms. *Proc. NSF/DoE Lake Tahoe Workshop on Hierarchical Approximation and Geometrical Methods for Scientific Visualization*, october 2002.

[CGG+05] P. Cignoni, F Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Batched multi triangulation. *IEEE Visualization 2005*, pages 27–34, 2005.

[DFKP04] L. De Fioriani, L. Kobbelt, and E. Puppo. A survey on data structures for level-of-detail models. In N. Dogson, M Floater, and M. Sabin, editors, *Advances in Multiresolution for Geometric Modelling, Series in Mathematics and Visualization*, pages 75–88, 2004.

[DFMMP00] L. De Fioriani, P. Magillo, F. Morando, and E. Puppo. Dynamic view-dependent multiresolution on a client-server architecture. In *CAD Journal*, pages 805–823, 2000. Special Issue on Multiresolution Geometric Models.

[DFMP98] L. De Fioriani, P. Magillo, and E. Puppo. Efficient implementation of multi-triangulations. In *Visualization '98 Proceedings*, pages 43–50. IEEE, 1998.

[DHZ91] M.J. De Haemer and M.J. Zyda. Simplification of objects rendered by polygonal approximations. *Computer Graphics, 15(2)*, pages 175–184, 1991.

[DP02] C. DeCoro and J. Pajarola. Xfastmesh : Fast view-dependent meshing for external memory. *Proc. IEEE Visualization 2002*, pages 363–370, 2002.

[DWS+97] M Duchaineau, M. Wolinsky, D. Sigeti, M Miller, C. Aldrich, and M. Mineev-Weinstein. Roaming terrain : Real-time optimally adapting meshes. *Proc. IEEE Visualization 1997*, 1997.

[Ede99] Herbert Edelsbrunner. Simplical complexes. In *meeting 6*, 1999. http ://www.cs.berkeley.edu/ jrs/meshpapers/edels/.

[Gar99] Michael Garland. Multiresolution modeling : Survey and future opportunities. 1999.

[GH97] M. Garland and P. S. Heckbert. Surface simplification using quadrics error metrics. *Proc. SIGGRAPH 1997*, pages 209–216, 1997.

[GS02] M. Garland and E. Shaffer. A multiphase approach to efficient surface simplification. *Proc. IEEE Visualization 2002*, pages 117–124, 2002.

[GWH01] M. Garland, A. Wilmott, and P. S. Heckbert. Hierarchical face clustering on polygonal surfaces. *ACM Symposium on Interactive 3D Graphics*, march 2001.

[HDD+93] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. *TR 93-01-01, Dept. of Computer Science and Engineering, University of Washington*, January 1993.

[HG97] P. S. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. *Technical report, Carnegie Mellon University*, 1997.

[HHK+95] T. He, L. Hong, A. Kaufman, A. Varshney, and S. Wang. Voxel-based object simplification. In *Visualization '95*. IEEE Comput. Soc. Press, 1995.

[Hop96] Hugues Hoppe. Progressive meshes. *Computer Graphics Proceedings, ACM SIGGRAPH*, pages 99–108, 1996.

[Hop97] Hugues Hoppe. View dependent refinement of progressive meshes. *Computer Graphics Proceedings, ACM SIGGRAPH*, pages 189–198, 1997.

[Hop98] Hugues Hoppe. Smooth view-dependent level-of-details control and its application to terrain rendering. *Visualization '98 Proceddings*, pages 35–42, 1998.

[ILGS03] M. Isenburg, P. Lindstrom, S. Gumhold, and J. Snoeyink. Large mesh simplification using processing sequences. *Visualization '03*, pages 465–472, 2003.

[KK98] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. In *Journal of Parallel and Distributed Computing 48*, pages 98–129, 1998.

[KT96] A.D. Kalvin and R. Taylor. Superfaces : Polygonal mesh simplification with bounded error. *IEEE Computer Graphics and A, 16(3)*, pages 64–77, 1996.

[Lin00] Peter Lindstrom. Out-of-core simplification of large polygonal models. *Computer Graphics Proceedings, ACM SIGGRAPH*, pages 259–262, 2000.

[Lin03] Peter Lindstrom. Out-of-core construction and visualization of multiresolution surfaces. *Proc 2003 Symp. Interactive 3D Graphics*, pages 93–102, 2003.

[LPC+00] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital michelangelo project. *SIGGRAPH 2000*, pages 131–144, 2000.

[Lue01] D. P. Luebke. A developer's survey of polygonal simplification algorithms. *IEEE Computer Graphics and Applications, 21(3)*, pages 24–35, 2001.

[RB93] J Rossignac and P Borrel. Multi-resolution 3d approximations for rendering complex scenes. In B. Falcidieno and T. Kunii, editors, *Modeling in Computer Graphics : Methods and Applications*, pages 455–465, 1993.

[SCESL02] C. Silva, Y. Chiang, J. El-Sana, and P. Lindstrom. Out-of-core algorithms for visualization and computer graphics. *In Visualisation'02 Course notes*, 2002.

[SG01] E. Shaffer and M. Garland. Efficient adaptive simplification of massive meshes. *Proc. IEEE Visualization 2001*, pages 127–134, october 2001.

[SG05] E. Shaffer and M. Garland. A multiresolution for massive meshes. *Proc. IEEE Visualization 2005, vol 11*, pages 117–124, march/april 2005.

[SZL92] W. Shroeder, J. Zarge, and W. Lorensen. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH '92 Proceddings) 26*, 2 1992.

[Tur92] Greg Turk. Re-tiling polygonal surfaces. In *Computer Graphics (SIGGRAPH '92 Proceedings), volume 26*, pages 55– 64, July 1992.

[VBLT05] F. Vivodtzev, G.P. Bonneau, and P. Le Texier. Topology preserving simplification of 2d non-manifold. meshes with embedded structures. *The Visual Computer, Vol. 21, No. 8*, 2005.

[WK03] J. Wu and L. Kobbelt. A stream algorithm for the decimation of massive meshes. *Graphics Interface '03*, pages 185–192, 2003.

[XV96] J. Xia and A. Varshney. Dynamic view-dependent simplification for polygonal models. In *Visualization '96 Proceedings*, pages 327–334. IEEE, 1996.

[YSGM04] S. Yoon, B. Salomom, R. Gayle, and D. Manocha. Quick-vdr : Interactive view-dependent rendering of massive models. *Proceeding of the IEEE Visualization 2004*, pages 131–138, 2004.