

# Contributions au co-design de noyaux irréguliers sur architectures manycore : cas du remaillage anisotrope multi-échelle en mécanique des fluides numérique.

NNT: 2018SACLE012

Thèse de doctorat de l'Université Paris-Saclay préparée au Commissariat à l'énergie atomique et à l'Université d'Évry.

École doctorale 580 - Sciences et technologies de l'information et de la communication [STIC]. Spécialité : INFORMATIQUE.

par  
HOBY RAKOTOARIVELO

soutenue publiquement le 6 juillet 2018 devant le jury composé de :

- |  |                |
|--|----------------|
| ■ Pr. Hanna Klaudel , UNIVERSITÉ D'ÉVRY VAL D'ESSONNE        | [PRÉSIDENTE]   |
| ■ Pr. Jean-François Remacle , ÉCOLE POLYTECHNIQUE DE LOUVAIN | [RAPPORTEUR]   |
| ■ Dr. Frédéric Gava , UNIVERSITÉ PARIS-EST CRÉTEIL           | [RAPPORTEUR]   |
| ■ Dr. Laure Gonnord , UNIVERSITÉ LYON 1                      | [EXAMINATRICE] |
| ■ Dr. Adrien Loseille , INRIA SACLAY                         | [EXAMINATEUR]  |
| ■ Pr. Franck Pommereau , UNIVERSITÉ D'ÉVRY VAL D'ESSONNE     | [DIRECTEUR]    |
| ■ Dr. Franck Ledoux , COMMISSARIAT À L'ÉNERGIE ATOMIQUE      | [DIRECTEUR]    |

CONTRIBUTIONS AU CO-DESIGN DE NOYAUX IRRÉGULIERS SUR ARCHITECTURES MANYCORE.  
CAS DU REMAILLAGE ANISOTROPE MULTI-ÉCHELLE EN MÉCANIQUE DES FLUIDES NUMÉRIQUE.

**Résumé :** La simulation numérique d'écoulements complexes telles que les turbulences ou la propagation d'ondes de choc implique un temps de calcul conséquent pour une précision industrielle acceptable. Pour accélérer ces simulations, deux recours peuvent être combinés : l'adaptation de maillages afin de réduire le nombre de points d'une part, et le parallélisme pour absorber la charge de calcul d'autre part. Néanmoins réaliser un portage efficient des noyaux adaptatifs sur des architectures massivement parallèles n'est pas triviale. Non seulement chaque tâche relative à un voisinage local du domaine doit être propagée, mais le fait de traiter une tâche peut générer d'autres tâches potentiellement conflictuelles. De plus, les tâches en question sont caractérisées par une faible intensité arithmétique ainsi qu'une faible réutilisation de données déjà accédées. Par ailleurs, l'avènement de nouveaux types de processeurs dans le paysage du calcul haute performance implique un certain nombre de contraintes algorithmiques. Dans un contexte de réduction de la consommation électrique, ils sont caractérisés par de multiples cores faiblement cadencés et une hiérarchie mémoire profonde impliquant un coût élevé et asymétrique des accès-mémoire. Ainsi maintenir un rendement optimal des cores implique d'exposer un parallélisme très fin et élevé d'une part, ainsi qu'un fort taux de réutilisation de données en cache d'autre part. Ainsi la vraie question est de savoir comment structurer ces noyaux data-driven et data-intensive de manière à respecter ces contraintes ?

Dans ce travail, nous proposons une approche qui concilie les contraintes de localité et de convergence en termes d'erreur et qualité de mailles. Plus qu'une parallelisation, elle s'appuie une re-conception des noyaux guidée par les contraintes hardware en préservant leur précision. Plus précisément, nous proposons des noyaux locality-aware pour l'adaptation anisotrope de variétés différentielles triangulées, ainsi qu'une parallelisation lock-free et massivement multithread de noyaux irréguliers. Bien que complémentaires, ces deux axes proviennent de thèmes de recherche distinctes mêlant informatique et mathématiques appliquées. Ici, nous visons à montrer que nos stratégies proposées sont au niveau de l'état de l'art pour ces deux axes.

**Mots-clés :** noyaux irréguliers, adaptation anisotrope de maillages, calcul parallèle, machines manycore.

A CODESIGN APPROACH OF IRREGULAR KERNELS ON MANYCORE ARCHITECTURES.  
CASE OF MULTI-SCALE ANISOTROPIC REMESHING IN COMPUTATIONAL FLUID DYNAMICS.

**Abstract :** Numerical simulations of complex flows such as turbulence or shockwave propagation often require a huge computational time to achieve an industrial accuracy level. To speedup these simulations, two alternatives may be combined : mesh adaptation to reduce the number of required points on one hand, and parallel processing to absorb the computation workload on the other hand. However efficiently porting adaptive kernels on massively parallel architectures is far from being trivial. Indeed each task related to a local vicinity need to be propagated, and it may induce new conflictual tasks though. Furthermore, these tasks are characterized by a low arithmetic intensity and a low reuse rate of already cached data. Besides, new kind of accelerators have arised in high performance computing landscape, involving a number of algorithmic constraints. In a context of electrical power consumption reduction, they are characterized by numerous underclocked cores and a deep hierarchy memory involving asymmetric expensive memory accesses. Therefore, kernels must expose a high degree of concurrency and high cached-data reuse rate to maintain an optimal core efficiency. The real issue is how to structure these data-driven and data-intensive kernels to match these constraints ?

In this work, we provide an approach which conciliates both locality constraints and convergence in terms of mesh error and quality. More than a parallelization, it relies on redesign of kernels guided by hardware constraints while preserving accuracy. In fact, we devise a set of locality-aware kernels for anisotropic adaptation of triangulated differential manifold, as well as a lock-free and massively multithread parallelization of irregular kernels. Although being complementary, those axes come from distinct research themes mixing informatics and applied mathematics. Here, we aim to show that our devised schemes are as efficient as the state-of-the-art for both axes.

**Keywords :** irregular kernels, anisotropic mesh adaptation, parallel processing, manycore machines.



## **REMERCIEMENTS**



Je tiens en premier lieu à remercier personnellement les membres du jury, notamment :

- au PR. Jean-François Remacle et au DR. Frédéric Gava d'avoir, sans la moindre coersion, accepté la lourde tâche de relire et rapporter cette thèse, qui fut assez dense dans sa pré-version. Ce fut un honneur que d'avoir été rapporté par un des créateurs de GMSH, conscient de ce que représente le développement de codes de remaillage 3D et des subtilités inhérentes, ainsi qu'au spécialiste des algorithmes BSP conscient du temps de préparation et de post-traitement requis pour la sortie des résultats en calcul haute performance. Merci de votre reconnaissance quant à la quantité de travail accompli, ainsi que pour vos remarques judicieuses pour l'amélioration de la pré-version du manuscrit.
- au PR. Hanna Klaudel d'avoir endossé le rôle du président du jury, pour ses conseils et remarques lors des répétitions de la soutenance; ainsi qu'au DR. Laure Gonnord qui s'est déplacée exprès de Lyon, et au DR. Adrien Loseille spécialiste reconnu en adaptation anisotrope de maillages pour en avoir été les examinateurs.

Je tiens à exprimer ma sincère reconnaissance à mes deux encadrants, la super team des "Franck" :

- DR. Franck Ledoux pour une multitude de raisons que je ne puis énumérer de manière exhaustive. Je te suis vraiment reconnaissant de m'avoir pris en stage puis en thèse ainsi que de t'être démené pour mon postdoc. Je tiens en particulier à souligner tes qualités humaines : j'ai rarement connu des gens avec une telle capacité d'écoute et de compréhension, ainsi que ta disponibilité (malgré les 9 000 kms de distance durant une année).
- PR. Franck Pommereau, particulièrement pour la préparation de la soutenance. Merci pour ta bienveillance et tes conseils avisés qui ont fait qu'elle se soit très bien passée. Merci également pour ton partage d'expérience et les discussions qui m'ont aidé à mieux cerner le monde de la recherche publique, et en particulier des contraintes et règles implicites propres à la profession.

Je tiens également à remercier toutes les personnes qui m'ont aidé ou contribué, de quelque manière que ce soit, au bon déroulement de cette thèse. Merci :

- au DR. Cédric Chevalier pour les relectures des articles et ses remarques judicieuses pour les exposés en anglais; à Nicolas Le-Goff pour son aide quant au debugging et déploiement des codes sur les calculateurs, ainsi qu'au DR. Jean-Christophe Weill pour ses conseils avisés.
- à Isabelle Visotto, Brigitte Sadoule et Murielle Bourgeois pour leur aide pendant mes galères administratives. En plus d'une efficacité redoutable, elles sont également d'une gentillesse inouïe.
- à Thao Le-Chi et Denis Lubin d'avoir rendu l'antre des doctorants, connu aussi sous le nom de Teratec, comme un lieu accueillant et chaleureux. L'humour particulier de Denis me manquera sûrement.

\* \* \*

Ces trois années et demie de thèse ont été une longue aventure, avec certes ses hauts et ses bas, mais riche en expériences et en rencontres. Durant ce périple, j'ai eu la chance de cotoyer pas mal de personnes chaleureuses, avec qui j'ai gardé de bons souvenirs. Je tiens à les remercier en ces quelques lignes, notamment :

- ceux avec qui j'ai commencé mon aventure depuis le stage : DR. Gautier Dakin et DR. Rémi Barat. Merci pour votre aide et votre bonne humeur ainsi que ces bons moments passés ensemble. L'humour "un peu" salace et décapant de Gautier me manquera sûrement. Sinon je ne pardonnerai jamais à Rémi, ce gourou du sport, de m'avoir enrôlé au Viet-vo-dao (j'ai vraiment kiffé au fait) et tenté de faire de même pour l'escalade (ah non je ne me ferai pas avoir).
- les anciens de la tribu, j'ai nommé DR. Sébastien Morais et DR. Xavier Valentin. J'ai kiffé avoir assisté au concert d'IAM avec vous deux. Bon j'espère que Séb prend maintenant en compte les comms. Sinon je retiendrai toujours cette image de Xavier esquissant une série d'équations tout en écoutant et en rappant du NTM : c'est juste stylé.
- la team actuelle guidée par notre vénérable gourou Guillaume Morel, animée par Éloïse Billa et Théo Saillant, ainsi que les bébés thésards Nestor Demeure et Christina Paulin (son encadrante, mdr) pour la super ambiance dans le bureau. Je retiendrai notamment les "*j'ai faim !! j'en ai marre !!*" répétitifs d'Éloïse, les débats enflammés avec Théo, la minute culture gé. avec Nestor, sans oublier la générosité sans pareil de Christina, notre fournisseur officiel de chocolats suisses. Idem pour les futurs docteurs et compagnons de galère de rédaction, j'ai nommé les trolls Hugo Taboada, Hugo Brunie, sans oublier Arthur Loussert. Je retiendrai votre enthousiasme ainsi que nos discussions trollesques qui ont rendu cette fin de thèse beaucoup moins morose.
- la team de la fac constitués des futurs doct.eurs.oresse Jéremy Sobéria, Johan Arcile, Célia Biane et Ludovic Platon. Merci en particulier à Jerem' pour ces innombrables fois où tu m'as aidé quand j'avais des trucs administratifs à faire et que je ne pouvais pas passer au labo. J'ai gardé un bon souvenir de l'école de printemps à Porquerolles, avec entre autres Johan et Ludo : qu'on se le dise, ça s'apparentait plus à une semaine de vacances que de formation.
- les stagiaires, et en particulier Aurélien Darivon (a.k.a. KIKI), Clément Chahbazian et Baptiste Deligny qui ont grandement contribué à l'ambiance chaleureuse dans le bureau, sans oublier Simon Calderan, et d'autres dont j'ai malheureusement oublié les noms.

Je remercie également mes potes de toujours Charles, Kévin, Matthias, Florian, Terry et Amélie pour ces bons moments passés ensemble, et qui m'ont permis de décompresser et de me déconnecter de la thèse quand j'avais la tête trop dans le guidon. Je tiens également à remercier ma famille pour leur soutien, et particulièrement ma tante Josette pour d'innombrables raisons. J'ai une pensée particulière pour ma mère, j'aurais tellement souhaité qu'elle soit là pour partager cette joie. Enfin merci infiniment à toi Julie de m'avoir épaulé et supporté durant tout ce temps.

\* \* \*



## TABLE DES MATIÈRES

### Introduction

Contexte et problématique . . . . .	8
Simulation numérique . . . . .	8
Problème de passage à l'échelle . . . . .	9
Boucle numérique adaptative . . . . .	11
Émergence des machines manycore . . . . .	12
Problématique . . . . .	14
Solution proposée . . . . .	15
Difficultés et contributions . . . . .	15
Publications . . . . .	16
Structure du manuscrit . . . . .	16

### I NOTIONS ET MÉTHODES

#### 1 Notions préliminaires.

1.1 Notions de topologie . . . . .	19
1.1.1 Maillages et triangulations . . . . .	20
1.1.2 Variétés, atlas et orientabilité . . . . .	21
1.1.3 Représentation et stockage . . . . .	25
1.2 Notions de parallélisme . . . . .	28
1.2.1 Architectures de calcul . . . . .	28
1.2.2 Modèles de parallélisme . . . . .	31
1.2.3 Évaluation pratique . . . . .	33

#### 2 Adaptation de surfaces triangulées.

2.1 Extraction de la surface . . . . .	35
2.1.1 Cas d'un volume discret . . . . .	36
2.1.1.1 Discrétisation du volume . . . . .	37
2.1.1.2 Extraction de la surface . . . . .	38
2.1.2 Cas d'un volume implicite . . . . .	39
2.1.2.1 Extraction de la surface . . . . .	39
2.1.2.2 Résolution des ambiguïtés . . . . .	40
2.1.2.3 Stratégies accélératrices . . . . .	40
2.2 Adaptation de la surface . . . . .	42
2.2.1 Anisotropie et qualité d'un maillage . . . . .	42
2.2.1.1 Impact sur les solveurs numériques . . . . .	42
2.2.1.2 Forme et alignement optimal . . . . .	43
2.2.2 Approches basées sur une paramétrisation . . . . .	45
2.2.3 Approches variationnelles . . . . .	45
2.2.3.1 Répartition des points . . . . .	46
2.2.3.2 Noyaux globaux . . . . .	47
2.2.4 Approches locales . . . . .	48
2.2.4.1 Noyaux locaux . . . . .	48
2.2.4.2 Reconstruction de la surface . . . . .	49
2.2.4.3 Cartes de tailles . . . . .	50
2.3 Synthèse . . . . .	51

### II CONTRIBUTIONS

#### 3 Design de noyaux surfaciques locality-aware.

3.1 Introduction . . . . .	53
----------------------------	----

54

3.1.1	Cadre et contraintes . . . . .	54
3.1.2	Démarche et contributions . . . . .	55
3.2	Design des noyaux . . . . .	56
3.2.1	Hypothèses et optimalité . . . . .	56
3.2.2	Pré-traitement . . . . .	57
3.2.2.1	Extraction des ridges . . . . .	58
3.2.2.2	Représentation de la topologie . . . . .	58
3.2.2.3	Orientation de la surface . . . . .	60
3.2.3	Reconstruction et projection . . . . .	61
3.2.3.1	Patch spline par maille . . . . .	62
3.2.3.2	Blending pour la continuité $G^1$ . . . . .	63
3.2.3.3	Notre opérateur de projection . . . . .	64
3.2.4	Recherche locale . . . . .	66
3.2.4.1	Raffinement . . . . .	66
3.2.4.2	Simplification . . . . .	67
3.2.4.3	Relaxation . . . . .	69
3.2.4.4	Lissage . . . . .	70
3.2.5	Stratégie complète . . . . .	78
3.3	Extension au cas anisotrope . . . . .	80
3.3.1	Répartition anisotrope des points . . . . .	80
3.3.1.1	Intérêt et principe . . . . .	80
3.3.1.2	Normalisation . . . . .	82
3.3.1.3	Gradation . . . . .	84
3.3.2	Transport de métriques . . . . .	87
3.4	Évaluation numérique . . . . .	91
3.4.1	Cadre, but et mesures . . . . .	91
3.4.2	Précision des noyaux . . . . .	92
3.4.3	Convergence . . . . .	96
3.5	Conclusion . . . . .	98
<b>4</b>	<b>Portage des noyaux sur les architectures manycore.</b>	<b>99</b>
4.1	Introduction . . . . .	100
4.1.1	Cadre et contraintes . . . . .	100
4.1.2	Irrégularité des noyaux . . . . .	101
4.1.3	Démarche et contributions . . . . .	103
4.2	Parallélisation des noyaux . . . . .	104
4.2.1	Extraction du parallélisme . . . . .	104
4.2.1.1	Notre approche . . . . .	105
4.2.1.2	Pour la simplification . . . . .	107
4.2.1.3	Pour la relaxation . . . . .	109
4.2.2	Restructuration des accès-mémoire . . . . .	111
4.2.2.1	Refonte en vagues synchrones . . . . .	111
4.2.2.2	Insertions de mailles . . . . .	112
4.2.2.3	Réduction de données . . . . .	113
4.2.3	Consistance des données d'incidence . . . . .	114
4.2.3.1	Contraintes en lock-free . . . . .	115
4.2.3.2	Notre synchronisation . . . . .	116
4.3	Évaluation numérique . . . . .	117
4.3.1	Cadre, architectures et paramètres . . . . .	117
4.3.2	Surcoûts et efficience de nos briques . . . . .	120
4.3.2.1	Extraction de tâches . . . . .	120
4.3.2.2	Notre synchronisation . . . . .	122
4.3.3	Scaling et surcoûts . . . . .	122
4.3.4	Débits de tâches par noyau . . . . .	125
4.4	Conclusion . . . . .	126

<b>Conclusion</b>	<b>128</b>
Approche et contributions . . . . .	128
Travaux annexes . . . . .	129
Limites et perspectives . . . . .	130
 III ANNEXE	
<b>Annexe A Noyaux numériques</b>	<b>133</b>
A.1 Reconstruction de la surface . . . . .	133
A.1.1 Paramétrisation quartique . . . . .	133
A.1.2 Jacobienne et aire . . . . .	133
A.2 Champ de tenseurs métriques . . . . .	134
A.2.1 Reconstruction de la hessienne . . . . .	134
A.2.2 Reconstruction des courbures . . . . .	137
A.2.3 Couplage des deux champs . . . . .	138
A.2.4 Interpolation des tenseurs . . . . .	139
<b>Annexe B Étude d'algorithmes pour l'extraction de tâches</b>	<b>140</b>
B.1 Extraction de stable maximal . . . . .	141
B.1.1 En séquentiel . . . . .	141
B.1.2 Cas des méthodes gloutonnes. . . . .	142
B.1.2.1 Approches probabilistes . . . . .	142
B.1.2.2 Approches spéculatives . . . . .	143
B.1.3 Profiling et analyse expérimentale . . . . .	145
B.1.3.1 Paramètres de tests . . . . .	145
B.1.3.2 Sensibilité à l'ordonnancement . . . . .	147
B.1.3.3 Sensibilité au placement mémoire . . . . .	148
B.1.3.4 Ratio de conflits . . . . .	149
B.1.3.5 Qualité de la solution . . . . .	149
B.1.4 Synthèse . . . . .	151
B.2 Extraction d'un couplage maximal . . . . .	151
B.2.1 Cadre et motivations . . . . .	151
B.2.2 Cas d'un graphe non-biparti. . . . .	152
B.2.3 Synthèse . . . . .	154
<b>Références</b>	<b>156</b>
Maillages et tessellations 3D . . . . .	156
Extraction d'isosurfaces et visualisation . . . . .	157
CFD, remaillage variationnel et anisotrope . . . . .	159
Reconstruction, discréétisation et optimisation de surfaces . . . . .	161
Architectures parallèles et applications . . . . .	164
Algorithmes d'approximation en optimisation combinatoire . . . . .	165
Algorithmes de graphes sur architectures manycore . . . . .	166

## **INTRODUCTION**

Le but de cette thèse est de présenter une manière de concevoir des algorithmes numériques irréguliers qui soit spécifiquement adaptée aux accélérateurs ou processeurs manycore sur un cas concret. La particularité de cette approche, c'est qu'elle ne repose pas que sur une parallélisation : le choix et la construction même des noyaux numériques est guidé par les contraintes hardware. D'où le terme "co-design" dans l'intitulé. À cette fin, nous prenons les noyaux d'adaptation anisotrope de maillages impliqués dans les simulations numériques en mécanique des fluides comme cas d'étude.

### **Contexte et problématique**

#### **Simulation numérique**

**CONCEPT.** Notre cas d'étude s'inscrit dans le contexte des simulations numériques en mécanique des fluides tels que les écoulements ou la propagation d'ondes de choc sur un domaine 3D. La simulation vise à prédire ou reproduire ces phénomènes par le calcul. C'est une alternative viable aux tests réels quand ces derniers sont financièrement coûteux (comme la fabrication et les tests de coques d'avions par exemple, voir figure 1), ou juste impossible à réaliser (mesures de l'intensité d'une déflagration nucléaire par exemple). En fait, elle est intensivement utilisée aussi bien dans la recherche scientifique que dans l'industrie. En effet,

- elle permet d'analyser et de voir des phénomènes inobservables (comme la formation de trous noirs) ou à des échelles qui sont hors de notre portée (distance en années-lumière, pas de temps en millions d'années, densité en milliards de tonnes par mm<sup>3</sup>) grâce au calcul haute performance.
- elle permet de prototyper et de tester rapidement des produits (typiquement l'usure et les contraintes appliquées sur les pièces en industrie aéronautique par exemple). Ainsi elle permet d'accélérer la chaîne de production depuis le design jusqu'à la validation du produit, en passant par les itérés de tests et de modification.

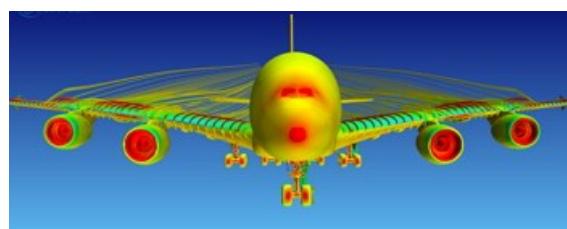


Figure 1: Calcul d'écoulements d'air sur un avion pour les tests d'aérodynamisme [ANSYS].

**PRINCIPE.** De manière concrète, le phénomène étudié est décrit par des équations différentielles<sup>1</sup> en vue d'une résolution par le biais de ce qu'on appelle un schéma numérique<sup>2</sup>. Ce dernier permet de résoudre de manière discrète et approchée une équation définie sur un domaine continu. Le schéma numérique est ensuite traduit en algorithme afin d'être exécuté sur un ordinateur. Pour comprendre la procédure, considérons l'exemple donné à la figure 2. Supposons que l'on dispose d'une pièce mécanique et que l'on veuille faire un calcul de contraintes dessus. Pour cela, la première étape serait de modéliser géométriquement la pièce en question à l'aide d'un logiciel de CAO<sup>3</sup>). Ensuite, la pièce est discrétisée en éléments géométriques simples appelées mailles, dont le tout forme ce qu'on appelle un maillage. Ensuite, on fixe les conditions initiales du problème (conditions aux bords par exemple) sur le maillage. Ici, on précise par exemple qu'un bord de la pièce est fixé sur un mur et que l'on exerce une traction sur l'extrémité de la pièce. Ensuite, l'équation est résolue sur le maillage par un solveur

<sup>1</sup>Par des équations aux dérivées partielles précisément, comme celles d'Euler ou de Navier-Stokes pour la simulation d'écoulements en mécanique des fluides.

<sup>2</sup>Par la méthode des différences finies, des éléments finis ou des volumes finis typiquement.

<sup>3</sup>CAO pour *conception assistée par ordinateur* dont le marché de solutions logicielles est disputé par de grands groupes telles qu'Autodesk, Ansys, Dassault systèmes, Siemens PLM software.

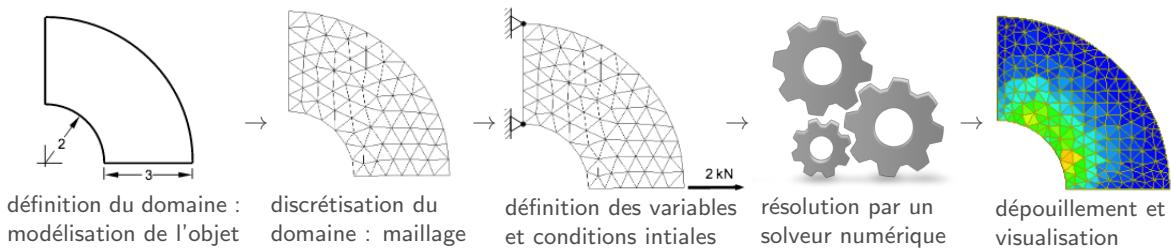


Figure 2: Exemple de workflow en simulation numérique.

numérique. Les résultats peuvent ensuite être dépouillées et visualisées. Pour cela, les grandeurs physiques calculées sont souvent représentées par des gradients de couleurs. Enfin, il est important de noter que la complexité algorithmique<sup>4</sup> et donc le temps de restitution de la simulation est directement liée au nombre de points (ou résolution) du maillage : plus on a de points, plus le temps de calcul sera important. Elle dépend aussi du type et de la précision du schéma numérique utilisé : plus celui-ci est précis (dit d'ordre élevé), plus les calculs sur et au voisinage des points (stencils) seront importants<sup>5</sup>.

### Problème de passage à l'échelle

CAS CONCRET. En fait, on va s'intéresser particulièrement aux simulations 3D en mécanique des fluides numérique, tel que le calcul d'écoulements illustré à la figure 1. En effet, ce sont les types de simulations que l'on retrouve dans l'industrie de pointe tels que les secteurs de l'énergie, l'aéronautique ou encore le spatial. Le problème vient du fait qu'elles impliquent un temps de restitution conséquent pour une précision acceptable en contexte industriel : c'est typiquement le cas pour les écoulements complexes telles que les turbulences ou la propagation d'ondes de choc à échelle réelle par exemple. Pour se fixer les idées, on va considérer l'exemple de la prédition du bang sonique en aéronautique décrite dans [59]. Il s'agit de l'onde de choc qui est générée quand un avion supersonique franchit le mur du son comme illustré à la figure 3. Ainsi, le but de la simulation sera de prédire la propagation de cette onde de choc. Sur cet exemple, une onde de choc est créée à 10km d'altitude par les irrégularités géométriques d'un avion (avec une taille caractéristique de l'ordre de 40m), et c'est son intensité au sol, à 10km plus bas, qui doit être minimisée comme illustré sur la figure 3.

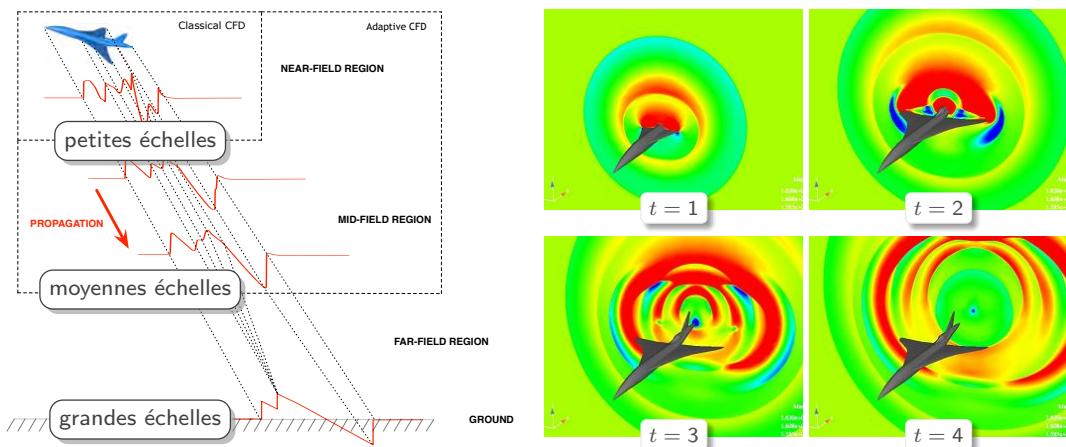


Figure 3: Exemple de la prédition du bang sonique en aéronautique [59].

<sup>4</sup>Il s'agit du coût théorique de l'algorithme c'est-à-dire le nombre d'instructions traitées en fonction de la taille  $n$  de l'instance du problème.

<sup>5</sup>Pour un schéma numérique implicite, on a une matrice carrée de taille  $n$  à inverser tandis que pour un schéma explicite, le pas de temps dépend du pas d'espace qui définit la finesse du maillage : ainsi le nombre d'itérés dépend indirectement de  $n$ , où  $n$  désigne le nombre total de degrés de liberté du problème.

Le problème avec ce genre de simulations est qu'elle fait intervenir plusieurs échelles. Ici, on a typiquement des échelles de l'ordre (1) du mètre près de l'avion, (2) du centaine de mètres plus loin, et enfin (3) de l'ordre du kilomètre près du sol. Ainsi il faudrait avoir une résolution très fine de maillage et donc un nombre important de points pour pouvoir capturer correctement les phénomènes à toutes les échelles (front de choc ainsi que les oscillations). Par conséquent, cela implique un nombre important de points et donc un temps de calcul conséquent comme illustré sur la table 1.

Table 1: Statistiques sur l'exemple de la figure 3.

itér.	points	tétrahédres	durée <sup>a</sup>
5	432 454	2 254 826	1h10'
10	608 369	3 294 197	2h54'
15	1 104 910	6 243 462	6h09'
20	1 757 865	10 125 724	11h15'
25	2 572 814	14 967 820	18h47'
30	<b>3 299 367</b>	19 264 402	<b>28h35'</b>

<sup>a</sup>mesures sur une machine 8-core classique.

CONTRAINTEs. En fait, on est confronté à un défi majeur en mécanique des fluides numérique : avoir des simulations plus réalistes avec un temps de restitution raisonnable. Pour être moins dépendant des hypothèses physiques simplificatrices, le focus est mis sur les schémas numériques. Ainsi, ces dernières deviennent de plus en plus précis mais complexes<sup>6</sup>, ce qui a pour effet d'augmenter considérablement le temps de calcul. À titre d'exemple concret, l'évolution des types de simulations effectuées chez AIRBUS est donnée à la figure 4. Ici, la charge de calcul induite par les tests est donnée sur l'axe de gauche, tandis que la puissance de calcul requise est décrite par l'axe de droite. Au début des années 90, ils se reposent entièrement sur des modèles de turbulences visant à simplifier les équations de NAVIER-STOKES pour simuler les tourbillons présents dans la traînée des avions (RANS). À l'horizon 2020 à 2030, l'objectif sera de ne recourir à ces modèles simplificateurs que pour les tourbillons de petites tailles, ceux de grande taille seront directement calculés tels quels (LES), voire ne recourir à aucun modèle de turbulence du tout (DNS). Le problème est que la puissance de calcul disponible ne suit pas. En fait, il faudrait disposer de calculateurs exaflopiques, c'est-à-dire capables d'effectuer  $10^{18}$  opérations en virgule flottante par seconde, pour pouvoir réaliser de telles simulations. Or on ne dispose actuellement que de machines petaflopiques ( $10^{15}$  flops) dans les centres de calcul haute performance.

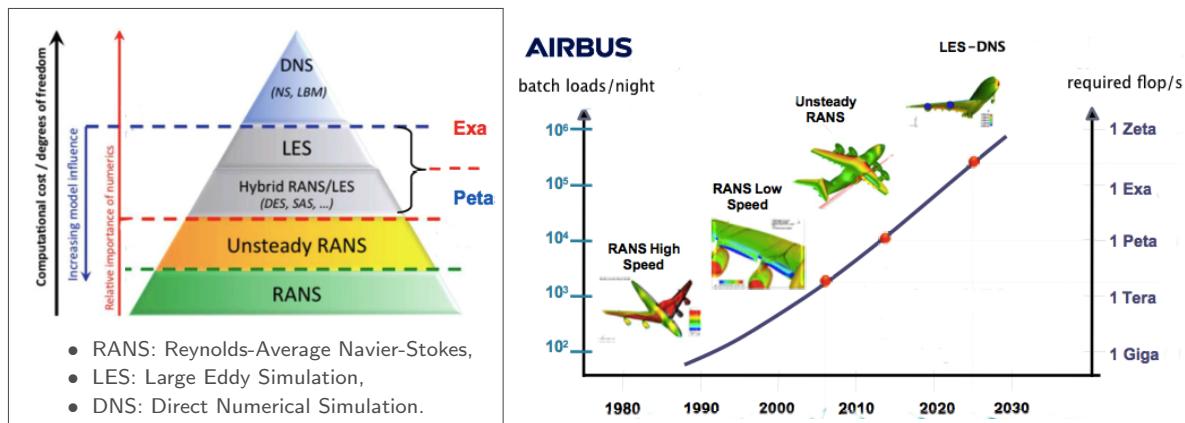


Figure 4: Évolution des simulations en aéronautique et puissance de calcul requise [67, 162].

<sup>6</sup>Il s'agit de schémas d'ordre élevé faisant intervenir des termes et dérivées d'ordre supérieur dans les approximations des fonctions. Cela complexifie la résolution numérique des équations et implique un temps de calcul plus important.

### Boucle numérique adaptative

ALTERNATIVES. In fine, le temps de calcul prohibitif s'explique principalement du fait que les schémas numériques ont besoin d'un nombre important de points afin de capturer finement toutes les échelles de la solution physique calculée sur le domaine. Avec la montée en ordre en plus, ces schémas deviennent de plus en plus complexes et coûteux ce qui rallonge encore plus le temps de calcul. Pour réduire ce temps, plusieurs recours existent dont :

- l'adaptation de maillages : elle vise à réduire le nombre de points requis et donc le temps de restitution en adaptant la finesse du maillage et l'orientation des mailles à l'erreur de la solution calculée sur ce domaine. Elle fait intervenir un solveur numérique ainsi qu'un remailleur 3D.
- le calcul parallèle avec décomposition de domaines : elle vise à absorber la charge de calcul en découplant les tâches de manière à pouvoir être traitées en parallèle sur une machine multi-processeurs à mémoire distribuée ou partagée.

En fait, ces deux techniques sont souvent combinées en pratique afin de tirer profit de leurs avantages respectifs. Dans le cadre d'une simulation numérique adaptative, la boucle de calcul entrelace les phases de résolution et de re-discretisation comme illustré sur la figure 5. En modifiant graduellement la discréttisation du domaine à chaque pas de temps, cela permet d'équi-répartir l'erreur de la solution numérique calculée sur ce domaine tout en réduisant le nombre de points et donc le temps de restitution de la simulation. Pour cela, la solution est d'abord calculée sur le maillage initial. Ensuite, son erreur d'interpolation est estimée sur le maillage courant et on fait un test de convergence : si celle-ci cesse de décroître ou si on atteint un seuil d'erreur prescrit alors on arrête la simulation. Sinon une carte de densité de points est extraite à partir de l'estimation de l'erreur, et on procède à l'adaptation du maillage courant. Enfin, la solution calculée est transférée de l'ancien maillage vers le maillage adapté.

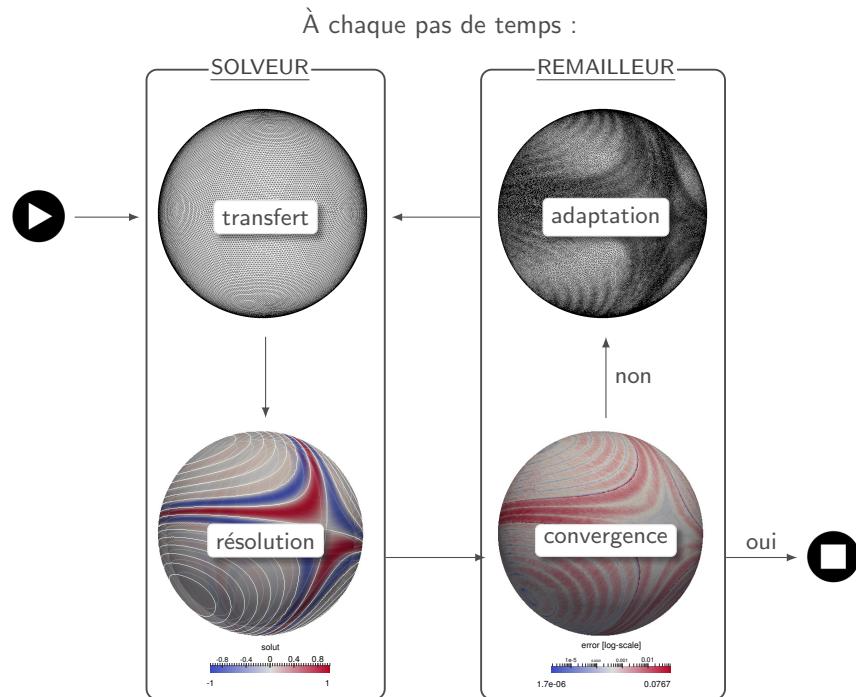


Figure 5: Principe de l'adaptation de maillage en simulation numérique.

POINT CRITIQUE. Notons que chaque phase de la boucle adaptative doit être parallélisée de manière efficiente pour obtenir une accélération substantielle, conformément à la loi d'AMDAHL [166]. La parallélisation efficiente de solveurs numériques est un problème bien étudié, notamment ceux impliquant la résolution de grands systèmes linéaires faisant intervenir des matrices creuses [163]. Il en est de

même pour le test de convergence et pour la phase de transfert de la solution. Par contre, l'adaptation efficiente de maillages sur architectures parallèles reste un problème non trivial. En fait, il s'agit d'un problème dit **irrégulier**, c'est-à-dire :

- **data-driven** dans le sens où les tâches ainsi que leurs dépendances varient dynamiquement et ne peuvent pas être inférées avant exécution. Ainsi le nombre de tâches ordonnancées à l'instant  $t$  ne peut être prédit statiquement. De plus, le fait de traiter une tâche peut invalider d'autres tâches déjà prévues;
- **data-intensive** dans le sens où la plupart des instructions sont dominées par des accès-mémoire, et souvent sur des données différentes. Ainsi, on a une faible réutilisation de données déjà accédées. De plus, il est difficile de maintenir un placement de données cache-aware, en raison des motifs d'insertion et de suppression de données qui sont non prédictibles.

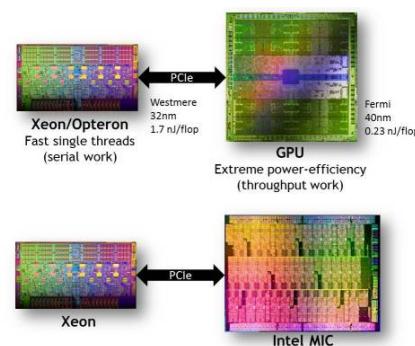
Ainsi pour maintenir une accélération substantielle de la boucle de calcul, la phase d'adaptation doit aussi être gérée efficacement : c'est le point sur lequel on va se focaliser dans cette thèse.

### Émergence des machines manycore

**CONTEXTE.** Nous avons vu à la figure 4 qu'il faudrait disposer de machines exaflopiques ( $10^{18}$  flops) pour réaliser des simulations de fluides à grande échelle (LES ou DNS), même en recourant à l'adaptation de maillages. En vue de répondre à ces besoins grandissants en puissance de calcul, les constructeurs de calculateurs à l'instar de Cray, Atos, IBM, Fujitsu, NEC se sont lancés dans une compétition mondiale pour franchir le cap de l'exaflop d'ici 2020. Pour se donner une idée concrète, cela revient à multiplier par 30 la capacité de calcul des machines les plus performantes, à l'instar de Tianhe-2 avec 33.9 pétaflops, ayant occupé la première place du Top500 de 2013 à 2016<sup>7</sup>. C'est un tournant et un défi majeur dans le paysage du calcul haute performance. En effet, il implique des investissements se chiffrant en millions de dollars<sup>8</sup>, mais également des contraintes technologiques relatives à la réduction de la consommation électrique induite par la multiplication des unités de calcul, typiquement de l'ordre du mégawatt. Un exemple de machine de classe exaflopique est donné à la figure 6. En cours de déploiement, le calculateur Tera-1000 est un cluster constitué de processeurs rapides Intel Xeon à 2.5 Ghz combinés avec des processeurs manycore Intel Knights Landing (72 cores à 1.4 Ghz) reliés par un réseau d'interconnexion à très faible latence BXI d'Atos.



(1) Tera-1000 du CEA (phase 1 sur 3)



(2) nœud de calcul hybride

Figure 6: Exemples de machine de classe exaflopique et type de nœud associé.

En fait, l'un des plus grands défis des constructeurs de calculateurs est de réduire la consommation électrique<sup>9</sup> (en watts) tout en maintenant une capacité de calcul élevée (en flops). Pour cela, il

<sup>7</sup> La liste des supercalculateurs les plus performants est répertoriée par le Top500, voir <http://www.top500.org>. Cette liste est mise à jour tous les 6 mois.

<sup>8</sup> Le département américain de l'énergie estime le budget minimum à 3 milliards de dollars pour y parvenir. Plus de 100 millions d'euros ont été investis par Atos en R&D à cette fin [source BFM Business].

<sup>9</sup> La puissance consommée par le processeur est  $P = P_d + V \times I_f$ , où  $P_d = C_e \times f \times V^2$  avec  $I_f$  le courant de fuite,  $C_e$  la capacité,  $V$  la tension et  $f$  la fréquence.

faudrait réduire la fréquence des processeurs, et donc augmenter le nombre de cores par noeud de calcul pour compenser. C'est ainsi qu'on assiste à l'émergence des accélérateurs matériels et des processeurs manycore. Ils sont caractérisés par un nombre conséquent de cores par noeud de calcul, mais une fréquence (1.5 GHz) et mémoire par core très limité comparé à des cores classiques de type XEON. Les calculateurs récents (notamment les têtes de liste du Top500) ont majoritairement adopté une architecture hybride avec des noeuds de calcul constitués de processeurs rapides (XEON ou OPTERON) couplés par des accélérateurs ou puces manycore (MIC, tilera, MPPA, ou GPU). L'évolution des architectures de processeurs dédiés au calcul haute performance et au deep learning est résumée à la table 2. En fait, l'application doit exposer un parallélisme explicite très élevé et très fin pour tirer profit de ces accélérateurs. D'un point de vue applicatif, c'est une énorme contrainte. En effet, elle implique de restructurer les algorithmes en une multitude de noyaux de calcul très simples et très locales pour maintenir une accélération substantielle sur ces cores faiblement cadencés.

Table 2: Évolution des processeurs en HPC.

sortie	architecture	cores <sup>a</sup>	GHz <sup>b</sup>	instr. <sup>c</sup>
2011	Tilera Tile-Gx	100	1.2	RISC
2012	Intel MIC Knights Corner	60	1.2	RISC
2016	Nvidia Tesla Kepler	2 280	1	RISC
2014	Kalray MPPA Bostan	256	0.4	RISC
2015	Kalray MPPA Coolidge	1 024	1	RISC
2015	Nvidia Tesla Maxwell	3 272	1.2	RISC
2015	Adapteva Epiphany-V	1 024	1	RISC
2016	Intel MIC Knights Landing	72	1.5	CISC
2016	Nvidia Tesla Pascal	3 840	1.4	RISC
2017	Intel MIC Knights Mill	72	1.5	CISC
2018	Nvidia Tesla Volta	6 048 <sup>d</sup>	1.4	RISC

<sup>a</sup>Les données sont celles d'un seul socket ou d'une seule carte accélératrice.  
<sup>b</sup>Il s'agit de la fréquence maximale d'un seul core.  
<sup>c</sup>Les jeux d'instructions du CPU peuvent être riches (CISC) ou réduits (RISC).  
<sup>d</sup>Ils regroupent 5 376 cores génériques et 672 cores dédiés au calcul tensoriel.

**POINT CRITIQUE.** En fait ce sont les accès de données qui limitent réellement le scaling sur ces architectures. En effet les applications sont de plus en plus gourmandes en données (**data-intensive**), or déplacer une donnée coûte plus cher en cycles qu'effectuer un calcul comme montré sur la figure 7. Cet aspect est encore plus critique lors des transferts de données du processeur vers la carte accélératrice (GPU ou MIC) qui passent par un bus lent de type PCI-express. Près de 43 % des instructions des codes de calcul sont des accès-mémoire, or l'amélioration des performances des RAM, en termes de bande-passante et latence, ne suit pas celle des processeurs (problématique du **memory-wall**). Pour pallier ce problème, les architectures récentes intègrent plusieurs niveaux de cache<sup>10</sup> dont la latence croît en fonction de la profondeur dans la hiérarchie mémoire. Le recours aux caches permet d'atténuer la latence en préchargeant les données à utiliser aux prochaines instructions : accéder à une donnée en cache L1 nécessite 1 à 4 cycles tandis qu'accéder à la mémoire peut nécessiter 140 à 400 cycles. Le problème induit par la multiplication des cores est que le bus mémoire devient rapidement saturé par les accès concurrents qui croissent de manière exponentielle : c'est une des limitations majeures des architectures multiprocesseurs symétriques (SMP). Pour remédier à cela, les constructeurs décident de séparer la mémoire à différents endroits et de la reliée à différents bus, ce qui a amené à l'avènement des machines dites **NUMA**<sup>11</sup>. Il s'agit d'une architecture dans laquelle la mémoire est physiquement scindée et reliée par un bus rapide (QuickPath Interconnect sur Intel et HyperTransport sur AMD) mais

<sup>10</sup>Il s'agit d'une mémoire très rapide mais de très faible capacité (de l'ordre de 32 Ko à 33 Mo maximum). De manière générale, les processeurs disposent de deux caches L1 par core dédiés aux données et instructions, suivi d'un cache L2 par paire de cores puis d'un dernier cache L3 par groupe de cores.

<sup>11</sup>NUMA pour Non-Uniform Memory Access : la latence des accès-mémoire varie selon qu'ils soient locaux ou distants.

est perçue comme une seule mémoire partagée par le biais d'un espace d'adressage unique. Dans ce cadre, la latence relative aux accès-mémoire devient inégale puisqu'elle varie selon qu'on accède à une région mémoire locale ou distante. En fait, bien que la mémoire soit virtuellement partagée, le cout d'un accès-mémoire varie selon qu'il soit local ou distant. L'architecture d'une machine NUMA ainsi que la latence relative à chaque instruction du processeur sont données à la figure 7. Notons le gap entre la latence (en cycles) d'une instruction de calcul comparé à un accès de données à chaque niveau de la hiérarchie.

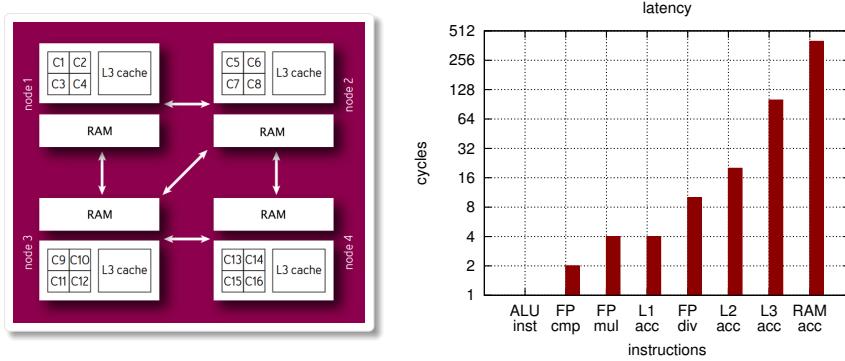


Figure 7: Architecture d'une machine NUMA et latence des instructions du processeur.

CONSÉQUENCES. D'un point de vue applicatif, il faudrait donc réutiliser au maximum les données déjà accédées (et donc en cache) et limiter les accès distants pour maintenir un bon scaling à nombre élevé de cores. À noter que les caches obéissent au principe dit de **localité** : ils gardent en priorité les données qui ont été récemment utilisées ou les données ayant des références mémoire proches de celles qui sont actuellement utilisées. Pour tirer profit, il est donc nécessaire de restructurer les algorithmes de manière à maximiser la réutilisation des données d'une part, mais aussi de rendre les accès-mémoire les plus locaux possibles d'autre part, particulièrement en contexte NUMA. Cela peut se faire par le biais de techniques de **cache tiling** ou de **prefetching**. Enfin, les accès-mémoire doivent être **coalescents** afin de minimiser la forte latence lors des transferts de données vers la carte accélératrice.

## Problématique

CONTRAINTEES. Notre but est de réduire le temps de restitution de simulations de fluides à grande échelle par une boucle numérique adaptative (figure 5) sur ces architectures massivement multithread. À cette fin, nous allons nous focaliser sur la phase d'adaptation qui en est le point critique.

- Le problème est qu'il s'agit d'application dite **IRRÉGULIÈRE** : le nombre de tâches ordonnanciables varie au cours de l'exécution, et le fait de traiter une tâche peut invalider d'autres déjà prévues. En raison de ces dépendances de tâches dynamiques et non prédictibles, il n'est pas trivial d'exposer un parallélisme constant et maximal à tout instant. De plus, ses instructions sont dominées par des accès de données avec un faible taux de réutilisation de données.
- Par ailleurs, nous avons vu que maintenir un bon scaling sur ces machines n'est pas trivial. D'une part, cela implique d'exposer un parallélisme très fin et élevé pour tirer profit du nombre conséquent de cores faiblement cadencés. D'autre part, il faudrait exposer une forte localité et un fort taux de réutilisation de données pour maximiser le rendement de caches. Cela est nécessaire pour atténuer la forte latence lors d'un accès-mémoire ou lors des transferts de données depuis une mémoire distante (NUMA) ou vers la carte accélératrice (MIC, GPU).

Ainsi le réel problème consiste à concilier les deux contraintes : compte-tenu de leur irrégularité intrinsèque, comment exposer des noyaux adaptatifs sensibles à la localité-mémoire tout en restant aussi efficaces que les noyaux de référence ?

## Solution proposée

**PRINCIPE.** Afin de concilier ces deux contraintes, nous abordons le problème par une approche dite de **co-conception** décrite à la figure 8. Son essence réside dans le choix et la construction même des noyaux séquentiels qui est pensé de manière à exposer une très forte localité malgré leur irrégularité. Ici au lieu de paralléliser le meilleur noyau séquentiel, nous préférons une heuristique ou un algorithme d'approximation qui fournit un résultat de qualité similaire mais qui respecte les contraintes induites par le hardware.



- une approche formellement prouvée pour le transport optimal de tenseurs métriques sur une surface. Elle permet de limiter la perte d'anisotropie induite par les interpolations répétées des tenseurs métriques associés à chaque point lors des créations ou déplacements de points.

Pour réaliser un portage efficient sur nos architectures cibles, nous proposons :

- une extraction explicite des tâches de chaque noyau par une formulation en graphe de leurs dépendances. À cette fin, nous proposons deux nouveaux algorithmes lock-free pour le calcul de stables et le couplage de graphes non-bipartis, à l'issue d'une étude expérimentale et comparative de [204–206] sur des instances de graphes irréguliers [207]. Nous montrons qu'ils sont réellement adaptés à nos noyaux en termes de nombre de tâches extraites et coût induit.
- une manière de structurer les noyaux qui permet l'insertion coalescente des données et d'atténuer l'impact de la latence sur les accès-mémoire non prédictibles.
- une synchronisation lock-free pour les mises à jour des données topologiques en contexte massivement multithread. Nous montrons qu'il minimise les transferts de données et donc le surcoût induit comparé à celle dans [199, 167].

## Publications

Les résultats présentés dans cette thèse ont fait l'objet de trois articles dans des actes de conférences avec comité de lecture [RLP16, RLP+17, RL18], et d'une communication orale au sein d'un congrès [LR18].

Table 3: Publications.

- [RLP16]. Fine-grained locality-aware parallel scheme for anisotropic mesh adaptation. Hoby Rakotoarivelo, Franck Ledoux and Franck Pommereau. 25<sup>th</sup> International Meshing Roundtable, 2016, Washington DC, USA.
- [RLP+17]. Scalable fine-grained metric-based remeshing algorithm for manycore-NUMA architectures. Hoby Rakotoarivelo, Franck Ledoux, Franck Pommereau and Nicolas Le-Goff. Euro-Par: 23<sup>rd</sup> International Conference on Parallel and Distributed Computing, 2017, Santiago de Compostella, Spain. [30% of acceptation]
- [RL18] Accurate manycore-accelerated manifold surface remesh kernels. Hoby Rakotoarivelo and Franck Ledoux. 27<sup>th</sup> International Meshing Roundtable, 2018, Albuquerque NM, USA.
- [LR18] Parallel surface mesh adaptation for manycore architectures. Franck Ledoux and Hoby Rakotoarivelo. Communication orale à MeshTrends 13<sup>th</sup> World Congress in Computational Mechanics, 2018, New-York NY, USA.

## Structure du manuscrit

Dans les deux premiers chapitres, nous introduisons les notions minimales de topologie et de parallélisme requises pour la compréhension de la thèse. Une synthèse des algorithmes utilisés en adaptation de surfaces triangulées est ensuite donnée pour introduire certaines notions spécifiques ainsi que pour illustrer les aspects importants au design des noyaux.

Dans le chapitre 3, nous présentons la démarche initiée dans le design des noyaux surfaciques sensibles à la localité mémoire. Après, une brève présentation de la problématique abordée et des contributions, nous présentons la structure et les briques constitutives des noyaux d'adaptations de surfaces aussi bien à l'erreur d'une solution numérique qu'à l'erreur de la surface elle-même. Pour cela, nous montrons la manière dont nous gérons l'extraction des données topologiques, la reconstruction de la surface, le choix et contrôle des noyaux de recherche locale. Ensuite nous avons étendu l'approche au cas anisotrope où l'adaptation est guidée par la solution numérique calculée. Nous montrons en particulier comment transporter les métriques ou graduer la densité des points en préservant l'anisotropie.

Dans le chapitre 4, nous présentons l'approche initiée pour le portage des noyaux sur les architectures manycore. Après avoir mis en évidence l'irrégularité des noyaux, nous présentons les solutions noyaux-spécifiques basées sur une formulation en graphes des dépendances de données. Ensuite une étude approfondie des heuristiques pour l'extraction des tâches en contexte massivement multithread a été menée en vue d'en inférer nos propres algorithmes. Enfin nous montrons la restructuration des noyaux en vue d'obtenir des patterns d'accès-mémoires coalescents, et d'inférer un schéma de synchronisation efficient, en termes de transferts de données, pour les mises à jour du graphe d'incidence.

Nous concluons enfin par une synthèse de l'approche et des contributions, ainsi que les limitations et les perspectives de ce travail.

#### REMARQUE

Étant donnée la variété des thèmes abordés, nous avons conçu chaque chapitre comme une entité indépendante traitant sa propre problématique. Ainsi chaque chapitre inclut une introduction, un état de l'art disséminé au fur et à mesure<sup>a</sup>, sa propre section résultats et une conclusion. Bien qu'ils forment un tout, les thèmes étudiés dans chaque chapitre sont suffisamment vastes et disjoints pour justifier ce partitionnement.

<sup>a</sup>ce qui permet de se comparer aux travaux existants pour un point particulier

\* \* \*

**Part I**

---

**Notions et méthodes**

# Notions préliminaires.

Dans ce chapitre, nous introduisons les notions de topologie impliquée dans les noyaux d'adaptation de maillages d'une part, et les notions de parallélisme impliquée dans le portage des noyaux sur les architectures cibles d'autre part.

1.1	Notions de topologie . . . . .	19
1.1.1	Maillages et triangulations . . . . .	20
1.1.2	Variétés, atlas et orientabilité. . . . .	21
1.1.3	Représentation et stockage . . . . .	25
1.2	Notions de parallélisme . . . . .	28
1.2.1	Architectures de calcul . . . . .	28
1.2.2	Modèles de parallélisme . . . . .	31
1.2.3	Évaluation pratique . . . . .	33

## 1.1 NOTIONS DE TOPOLOGIE

Ici, nous définissons les notions de maillages et triangulations. Ensuite, nous présentons les types de surfaces que nous prenons en charge dans nos algorithmes. Nous montrons également celles que nous ne prenons pas en charge en expliquant pourquoi. Enfin, nous montrons comment nous représentons et stockons la surface triangulée et sa topologie.

VOISINAGE. On commence par rappeler la notion de voisinage qui est utile pour d'autres définitions. Intuitivement, le voisinage d'un point regroupe tous les points qui sont à une certaine distance de lui. Formellement, le voisinage (ouvert ou fermé) d'un point est défini par la notion de boule (ouverte ou fermée). En notant  $\|\cdot\|$  une norme quelconque sur  $\Omega$ , on a :

- la boule fermée d'ordre  $k$  d'un point  $x$  d'un domaine  $\Omega \subseteq \mathbb{R}^n$  est définie par :

$$B_k^n(x) = \{y \in \Omega, \|y - x\| \leq k\}, \quad (1.1)$$

- la boule ouverte d'ordre  $k$  correspond à l'intérieur de  $B_k^n$ , et est définie par :

$$\overset{\circ}{B}_k^n(x) = \{y \in \Omega, \|y - x\| < k\}. \quad (1.2)$$

- le bord de  $B_k^n$ , notée  $\partial B_k^n$  est la sphère  $S_k^{n-1}$  de dimension  $n - 1$  définie par :

$$S_k^{n-1}(x) = \{y \in \Omega, \|y - x\| = k\}. \quad (1.3)$$

De manière générale, on note  $\overset{\circ}{\Omega}$  l'intérieur de tout  $\Omega \subset \mathbb{R}^n$ ,  $\partial\Omega$  son bord, et  $\overline{\Omega}$  sa fermeture. Le voisinage continu d'un point, défini par sa boule, permet à son tour de définir la notion de topologie sur un domaine (voir définition 1).

**Définition 1** (TOPOLOGIE). Soit  $\Omega \subset \mathbb{R}^n$  un ensemble, et  $\mathcal{N} : \Omega \rightarrow \mathcal{P}(\Omega)$  la fonction qui associe à  $p \in \Omega$  son voisinage  $\mathcal{N}(p)$ . On dit que  $\mathcal{N}$  est une topologie sur  $\Omega$ , si elle vérifie que :

- chaque  $p \in \Omega$  appartient à chacun de ses voisinages;
- la réunion et l'intersection de deux voisinages de  $p \in \Omega$  est aussi un voisinage de  $p$ ;
- tout voisinage  $N \in \mathcal{N}(p)$  inclut un  $M \in \mathcal{N}(p)$  tel que  $N$  est un voisinage de tout  $u \in M$ .

Le couple  $(\Omega, \mathcal{N})$  forme un espace topologique, et les éléments de  $\mathcal{N}$  sont appelés les ouverts de  $\Omega$ .

### 1.1.1 Maillages et triangulations

En simulation numérique, un maillage fournit une représentation discrète d'un domaine comme sur l'exemple de la figure 1.1. Ce support discret permet de visualiser ou résoudre numériquement des équations aux dérivées partielles, modélisant des phénomènes physiques continus.

**Définition 2** (MAILLAGE). Soit  $\Omega$  un domaine continu et borné de  $\mathbb{R}^d$ ,  $d \in \mathbb{N}$ .

Un maillage  $\mathcal{T}_h$  est une discréétisation de  $\Omega$  par des éléments géométriques finis appelés mailles, telle que l'ensemble des mailles forme une partition de  $\Omega$ .

TOPOLOGIE. Par ailleurs, un maillage d'un domaine peut être vu comme son recouvrement par une collection de "cellules" de dimension différentes et interconnectées. Une cellule de dimension  $k$  est un nœud si  $k = 0$ , une arête si  $k = 1$ , une face si  $k = 2 = d - 1$  et une maille sinon (voir figure 1.2). Intuitivement, la topologie d'un maillage  $\mathcal{T}_h$  désigne la connectivité des cellules qui la constituent. Conformément à la définition 1, le couple  $(\mathcal{T}_h, \mathcal{N})$  est un espace topologique discret, où  $\mathcal{N}$  correspond à la relation de voisinage de chaque cellule de  $\mathcal{T}_h$ . On vérifie aisément que l'intersection ou la réunion de deux cellules est soit vide soit un ensemble de cellules appartenant à  $\mathcal{T}_h$ . Selon la régularité de sa topologie, un maillage peut être structuré ou non, au sens de la définition 3.

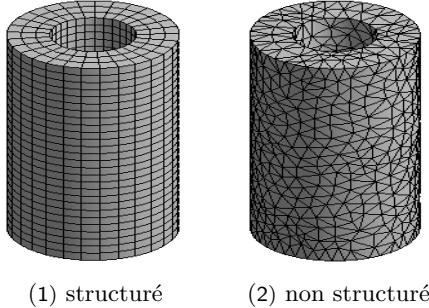


Figure 1.1: Classes usuelles de maillages<sup>1</sup>

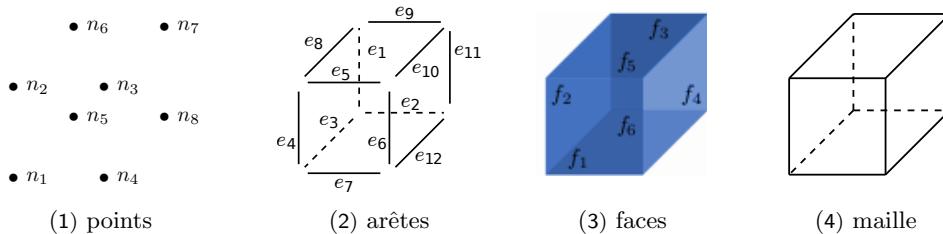


Figure 1.2: Décomposition d'une maille hexaédrique

**Définition 3** (MAILLAGE STRUCTURÉ). Un maillage est dit structuré si sa connectivité est régulière, autrement dit si le degré de chaque point est fixe. Les éléments d'un tel maillage sont généralement des quadrangles ou des hexaèdres.

Deux classes usuelles de maillages utilisées en simulation numérique sont représentées à la figure 1.1. Le premier représente un maillage *structuré*. Il permet de retrouver directement le voisinage de chaque point par adressage implicite, mais également d'obtenir une discréétisation à faible budget de points comparé à un maillage non structuré. Toutefois il est limité par la régularité du domaine, et nécessite souvent une décomposition préalable en blocs. Le second représente un maillage *non structuré* auquel

<sup>1</sup>source : <https://fr.wikipedia.org/wiki/Maillage>

cas la connectivité des points est irrégulière. Un tel maillage permet de discréteriser des domaines quelconques ou de capturer finement des écoulements sans directions privilégiés, mais requiert un stockage explicite de la topologie en mémoire. Il est généralement constitué de simplexes (triangles ou tétraèdres), mais peut contenir des éléments géométriques différents (prismes, pyramides etc.).

**Définition 4 (TRIANGULATION).** *Un triangulation  $\mathcal{T}_h$  d'un domaine  $\Omega \subset \mathbb{R}^3$  (ou de  $\partial\Omega$ ) est un maillage triangulaire de  $\Omega$  (ou de  $\partial\Omega$ ). Elle est dite conforme si l'intersection de deux mailles adjacentes se réduit à une arête.*

Trois exemples de triangulations invalides et non conformes sont illustrés à la figure 1.3 dans le cas planaire. La première est invalide car une partie de  $\Omega$  n'est pas recouverte, tandis que la seconde l'est en raison d'un chevauchement de mailles. La dernière est non conforme en raison de deux mailles  $K_1, K_2$  incidentes à une maille  $K_3$ . Dans le cadre de notre étude, on se restreint aux triangulations conformes selon la définition 4. De plus, il s'agit de la discréterisation la plus courante de modèles géométriques issus de scanners tomographiques (en imagerie médicale) ou d'une conception assistée par ordinateur.

**Définition 5 (MAILAGE INTERPOLANT).** *Une triangulation d'une surface est dite interpolante si ses points sont exactement sur cette surface.*

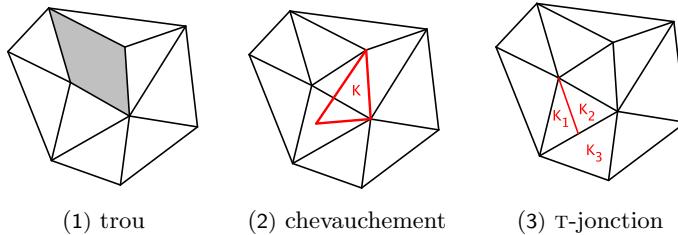


Figure 1.3: Exemple de triangulations invalides et non conformes.

### 1.1.2 Variétés, atlas et orientabilité.

Rappelons que notre but sera d'adapter des surfaces triangulées. Il y a plusieurs types de surfaces dont certaines ne sont pas trivialement discréterisables. Dans notre cas, nous nous limitons à des surfaces qui sont des variétés au sens de la définition. Elles couvrent néanmoins les surfaces que l'on peut retrouver dans l'espace usuel  $\mathbb{R}^3$ .

**Définition 6 (VARIÉTÉ).** *Une variété  $\Gamma$  de dimension  $n$ , ou  $n$ -variété, est un espace topologique  $E = (\Gamma, \mathcal{N})$  connexe localement homéomorphe à un disque de  $\mathbb{R}^n$ . Une variété de dimension 1 est une courbe, et une variété de dimension 2 est une surface. Intuitivement,  $\Gamma$  est une variété si le voisinage de tout point de  $\Gamma$  est localement euclidien mais ne l'est pas nécessairement globalement. Sauf précision particulière, on va uniquement considérer des 2-variétés, c'est-à-dire des surfaces.*

Dans le cas discret, une surface triangulée  $\mathcal{T}_h$  est une variété discrète si pour tout point  $p$  :

- chaque arête incidente à  $p$  est incidente à exactement une ou deux faces.
- le graphe d'adjacence des mailles incidentes à  $p$  est un cycle si  $p \in \Gamma$  et un chemin si  $p \in \partial\Gamma$ .

Des exemples de triangulations ne correspondant pas à des variétés discrètes sont données à la figure 1.4. Dans le premier cas, une arête est commune à trois faces. Dans le second cas, le voisinage d'un point à  $\Omega$  est incomplet dans le sens où l'ensemble des mailles incidentes à ce point ne constitue pas une boule fermée de  $\mathbb{R}^2$ . Autrement dit le graphe d'adjacence des mailles incidentes au point n'est pas un chemin. Dans le dernier cas, le voisinage d'un point interne n'est pas homéomorphe à un unique disque de  $\mathbb{R}^2$  puisqu'on peut obtenir deux boules fermées distinctes par "aplatissement".

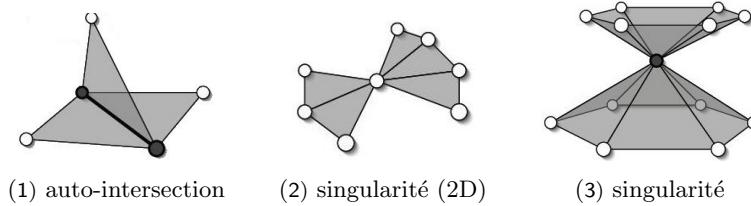


Figure 1.4: Exemples de triangulations qui ne sont pas des variétés discrètes [1].

EXEMPLE. Le globe terrestre est un exemple concret de variété tel qu'illusté à la figure 1.5. En effet, le voisinage de chaque point  $x$  de la surface  $S$  du globe est localement homéomorphe à une disque de  $\mathbb{R}^2$  (c'est-à-dire quelque chose de plat). Ce disque représente la *carte locale* de  $S$  en  $x$ . Si l'ensemble des cartes recouvre  $S$ , alors il constitue un *atlas* dont l'étude permet de rendre compte des propriétés d'une variété. D'un point de vue local, le plus court chemin entre deux villes est une ligne droite, mais globalement il s'agit de la géodésique (courbe) passant par ces deux villes. Sur la figure 1.5, les  $U_i$  correspondent aux ouverts de la variété, tandis que les  $\Omega_i$  sont les ouverts de  $\mathbb{R}^2$ . Les  $\varphi_i : U_i \rightarrow \Omega_i$  sont les homéomorphismes permettant d'associer chaque  $U_i$  à  $\Omega_i$ , et correspondent à la paramétrisation locale de la variété. Enfin les  $\varphi_{ij}$  sont des fonctions de transition permettant de "recoller" les cartes, également appelées applications de *changeement de cartes*.

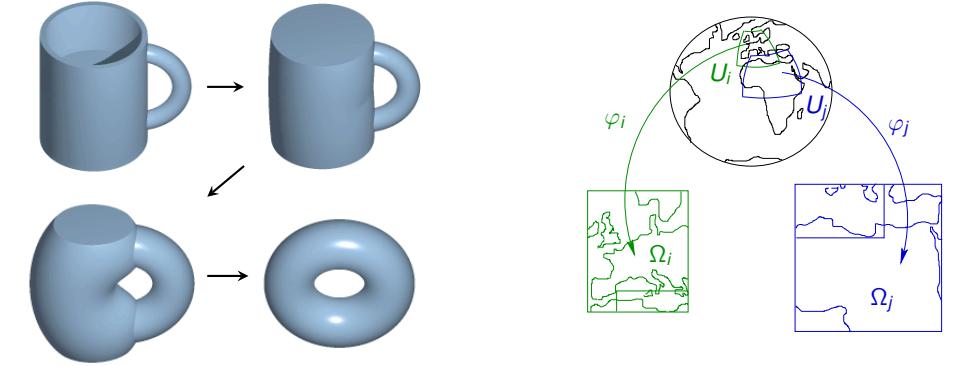


Figure 1.5: Exemples concrets d'homéomorphisme et variété de dimension deux.

**Définition 7 (ATLAS COMPATIBLES).** Un atlas est dit différentiable si les changements de cartes de toutes les intersections d'ouverts non vides le sont. Plus formellement, un atlas  $(U_x, \varphi_x)_{x \in \Omega}$  est dit de classe  $C^k$ ,  $k \in [1, \infty]$  si pour toute paire d'indices  $i$  et  $j$  telle que  $U_i \cap U_j = \emptyset$ , l'application de changement de cartes définie par  $\varphi_i \circ \varphi_j^{-1} : \varphi_j(U_i \cap U_j) \rightarrow \varphi_i(U_i \cap U_j)$  est un difféomorphisme de classe  $C^k$ . Deux atlas de classe  $C^k$  sont dits compatibles si leur réunion est aussi de classe  $C^k$ .

DÉRIVABILITÉ. La classe de variétés qui nous intéresse est celles des *variétés différentielles* au sens de la définition 8. Intuitivement, il s'agit de surfaces de  $\mathbb{R}^3$  qui sont dérivables. Formellement, ce sont de variétés que l'on peut munir d'un atlas différentiable dont les changements de cartes sont toutes de classe  $C^k$ . En remaillage surfacique, elle permet de plonger localement la variété dans le plan grâce aux cartes locales, puis de remailler la région à l'aide de noyaux 2D. Elle permet également de calculer l'orientation ou les courbures locales de la surface, ce qui est utile pour son remaillage direct.

<sup>2</sup>source: <https://fr.wikipedia.org/wiki/Homeomorphisme>.

**Définition 8** (VARIÉTÉ DIFFÉRENTIELLE). *Une variété différentielle de classe  $C^k$  est une variété munie d'une famille d'atlas de classe  $C^k$  mutuellement compatibles. Une variété est dite lisse si elle est de classe  $C^\infty$ .*

Si on considère une variété  $\Gamma$  comme un sous-espace de  $\mathbb{R}^n$  (on dit qu'elle est *plongée* dans  $\mathbb{R}^n$ ), alors elle peut posséder des points frontières qui vont constituer son bord  $\partial\Gamma$ . Dans ce cas, on dit que  $\Gamma$  est une variété à bord au sens de la définition 9), et le voisinage de tout point  $p \in \partial\Gamma$  déborde dans  $\mathbb{R}^n$ . Dans notre cas, nous ne gérons que des variétés sans bord (ou fermées). En effet, leur traitement complexifie les noyaux d'adaptation de maillages (un cas particulier de plus à gérer) sans apporter quelque innovation algorithmique.

**Définition 9** (VARIÉTÉ À BORDS). *Une variété à bord  $\Gamma$  de dimension  $n$  est un sous-espace topologique de  $\mathbb{R}^n$  dont les points admettent un voisinage homéomorphe à  $\mathbb{R}^n$  (point intérieur), ou bien à un ouvert de  $\mathbb{R}^n \times \mathbb{R}^+$  (point bordants). L'ensemble des points n'admettant que ce dernier type de voisinage constitue le bord de  $\Gamma$ .*

**ORIENTABILITÉ.** Afin de pouvoir représenter et stocker la surface triangulée, celle-ci doit être munie d'une orientation permettant de distinguer son intérieur de son extérieur. Comme illustré à la figure 1.6, toutes les variétés ne sont pas orientables. Dans notre cas, nous avons besoin d'une orientation consistante des mailles pour pouvoir calculer des quantités différentielles impliquées dans la création d'une carte de tailles d'arêtes, ou bien dans les noyaux (projection de points après un raffinement par exemple). L'orientation d'une variété dépend de la notion de *lacet* au sens de la définition 10.

**Définition 10** (LACET). *Un lacet  $\gamma$  d'une surface  $\Gamma$  est une courbe paramétrée  $\gamma : [0, 1] \rightarrow \Gamma$  telle que  $\gamma(0) = \gamma(1)$ . Ainsi, si on parcourt  $\gamma$ , le point de départ coïncide avec le point d'arrivée.*

Comme tout point d'une 2-variété  $\Gamma$  possède un voisinage homéomorphe à un disque de  $\mathbb{R}^2$ , choisir une orientation locale de  $\Gamma$  revient à choisir une orientation de ce disque, par exemple un repère local ou un sens de rotation. Intuitivement, une surface est orientable si lorsqu'on choisit une orientation en un point  $p$ , et que l'on déplace le long d'un lacet (au sens de la définition 10) en gardant le même choix d'orientation<sup>3</sup>, on retrouve la même orientation à l'arrivée qu'au départ. L'orientabilité d'une surface se définit ensuite à partir de la propriété de ses lacets à changer l'orientation ou non (définition 11). En effet, un lacet préserve l'orientation si l'orientation au départ et à l'arrivée sont identiques.

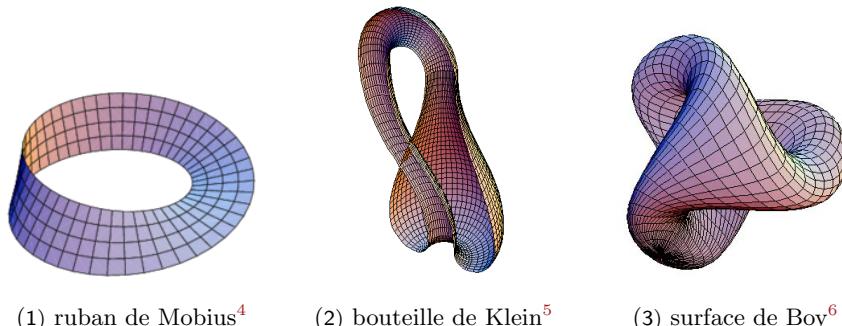


Figure 1.6: Exemples de variétés non orientables de  $\mathbb{R}^3$ .

**Définition 11** (VARIÉTÉ ORIENTABLE). *Une variété est orientable si tous ses lacets préservent l'orientation, sinon elle est juste dite non orientable.*

<sup>3</sup>grâce à la notion de *transport parallèle* que l'on définira plus tard.

<sup>4</sup>source : <https://da.wikipedia.org/wiki/Möbiusband>.

<sup>5</sup>source : <http://jeanne.allenfive.com/on-the-topic-of-topology-and-the-construction-of-the-klein-bottle/>

<sup>6</sup>source : <http://www.apprendre-en-ligne.net/blog/index.php/2009/01/27/1206-la-surface-de-boy>

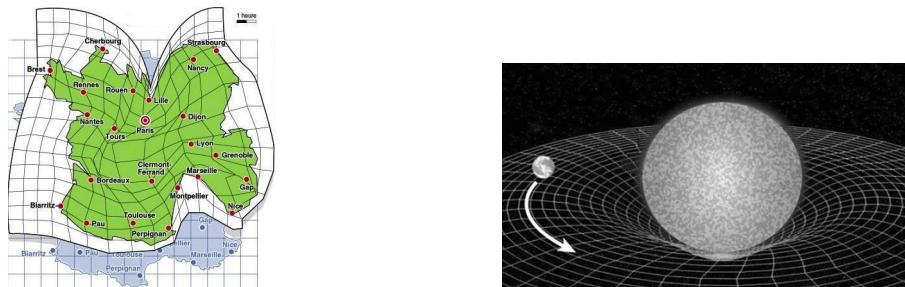
IN FINE. Dans notre cas, on se restreint aux surfaces triangulées de classe au moins  $C^2$  par morceaux, orientables et sans bord. La raison de cette contrainte d'orientabilité est double. En effet, une orientation consistante des mailles permet de définir un champ non ambigu de vecteurs normaux à la surface. Ce champ est utile pour la reconstruction discrète des quantités différentielles comme les courbures et directions principales, en tout point de  $\Gamma$ . Il permet également d'approcher localement  $\Gamma$  au voisinage d'un point ou sur une maille à l'aide d'une paramétrisation locale<sup>7</sup>. Par ailleurs, cette orientation nous permet de recourir à des structures de données basée sur un parcours ordonné de demi-arêtes (carte combinatoire par exemple, section 1.1.3), ou fournir des primitives topologiques basées sur un parcours barycentrique de mailles (localisation de points pour la projection de quantités ponctuelles par exemple).

DENSITÉ. Pour adapter la triangulation, nous avons besoin de savoir comment répartir les points sur la surface  $\Gamma$  et donc une densité sur les points de  $\Gamma$  (définition 12). Elle va redéfinir le voisinage continu de chaque point  $p$  du  $T_h$ , et donc la taille souhaitée des arêtes incidentes à  $p$ .

**Définition 12 (DENSITÉ).** *Intuitivement, une densité encode la répartition souhaitée des points sur la surface  $\Gamma$ . Formellement, c'est l'application  $\rho : \Gamma \rightarrow \mathbb{R}^+$  qui spécifie la taille du voisinage continu de tout point de  $\Gamma$ . Elle redéfinit la boule  $\{q \in \Gamma, \rho(p)\|q - p\|^2 \leq 1\}$  de tout point  $p \in \Gamma$ . Ainsi, la taille d'une arête  $[pq]$  est la longueur d'un segment géodésique sous-tendue par  $p$  et  $q$  conformément à  $\rho$  :*

$$\ell_h(pq) = \inf_{\gamma} \int_p^q \rho[s]^{\frac{1}{2}} \|\frac{d\gamma}{dt}[s]\| ds \quad (1.4)$$

En fait, quand on munit une variété  $\Gamma$  d'une carte densité  $\rho$  qui redéfinit le voisinage de chaque point de  $\Gamma$ , alors on est en train de construire une *variété riemannienne* au sens de la définition 13. En effet, on est en train de définir un espace métrique qui définit les distances localement sur chaque point conformément à  $\rho$ .



(1) une carte de la France selon le temps de trajet en TGV. La déformation résultante forme une variété riemannienne.<sup>8</sup>

(2) l'espace-temps est une variété pseudo-riemannienne: la terre se déplace en direction du soleil mais sa trajectoire est déviée.<sup>9</sup>

Figure 1.7: Exemples de variétés (pseudo)-riemanniennes.

**Définition 13 (VARIÉTÉ RIEMANNIENNE).** *Une variété riemannienne  $(\Gamma, g_p)$  est une variété  $\Gamma \subset \mathbb{R}^3$  munie d'un produit scalaire  $g_p : T_p \Gamma \times T_p \Gamma \rightarrow \mathbb{R}$  sur le plan tangent  $T_p \Gamma$  de tout point  $p \in \Gamma$ . Ainsi  $T_p \Gamma$  définit un espace métrique où la norme d'un vecteur  $\mathbf{v}$  est :  $\|\mathbf{v}\| = \sqrt{g_p(\mathbf{v}, \mathbf{v})}$ .*

Ainsi la surface  $\Gamma$  munie d'une carte de densité  $\rho$  peut être vue comme une variété riemannienne  $(\Gamma, g_p)$ , où  $g_p$  est le produit scalaire local au point  $p$  défini par :

$$g_p(\mathbf{u}, \mathbf{v}) = \rho(p) \langle \mathbf{u}, \mathbf{v} \rangle, \quad \forall \mathbf{u}, \mathbf{v} \in T_p \Gamma. \quad (1.5)$$

<sup>7</sup>Dans notre cas, la variété n'est connue qu'aux points de la triangulation. Ainsi elle peut être localement approchée par une surface quadrique, des splines, des patches de BEZIERS, ou encore des NURBS

<sup>8</sup>source: <http://neomansland.over-blog.org/article-20926040.html>.

<sup>9</sup>source: [http://www.uh.edu/~jclarage/astr3131/lectures/4/einstein/Einstein\\_stanford\\_Page7.html](http://www.uh.edu/~jclarage/astr3131/lectures/4/einstein/Einstein_stanford_Page7.html).

De manière générale, toute variété munie d'un produit scalaire local  $g_p$  est une variété riemannienne. Des exemples concrets de variétés (pseudo)-riemannniennes<sup>10</sup> sont donnés à la figure 1.7.

Cette notion de variété riemannienne est importante car elle va nous permettre de définir des noyaux uniques pour l'adaptation de triangulations à l'erreur d'une solution numérique ou à l'erreur de la surface elle même, en changeant juste la métrique  $g_p$  associée aux points  $p$  de la surface.

### 1.1.3 Représentation et stockage

Stocker le maillage implique de représenter et stocker sa topologie. En plus d'un stockage compact des points, arêtes ou mailles, la structure de données doit fournir les primitives d'accès et de modification des relations d'incidences en  $O(1)$ . Dans notre cas, c'est un aspect important car le stockage du maillage, ainsi que les motifs d'insertion et de mise à jour des données topologiques doivent tenir compte des contraintes hardware (préserver le placement mémoire, minimiser les indirections lors des requêtes de voisinage etc.). En réalité le choix d'une représentation repose sur quatre critères :

- le *domaine représenté* : avec ou sans bord, données associées ou non;
- le type de *requêtes* : accès en écriture ou non, type et degré de voisinage;
- le *stockage et accès aux données* : empreinte mémoire, mémoire partagée ou distribuée;
- la *généricité*.

La représentation choisie devrait idéalement permettre de vérifier les invariants topologiques avant validation des opérations effectuées. De nombreuses structures de données de maillages existent dans la littérature répondant plus ou moins à l'ensemble de ces critères. Cette section n'a pas pour but de les lister et de les comparer, mais plutôt de donner une synthèse des modèles de représentations et les structures de données qui en découlent. Elle s'appuie essentiellement sur les travaux décrits dans [2].

MODÈLES EXPLICITES. Ici, la topologie est vue comme une connexion de cellules différentes (points, arêtes ou mailles). Elle dépend de la dimension du maillage et est représentée par un graphe d'incidence au sens de la définition 14.

**Définition 14** (GRAPHE D'INCIDENCE). Soit  $\mathcal{T}_h$  un maillage de dimension  $d$ .

Le graphe d'incidence associé à  $\mathcal{T}_h$  décrit les connectivités des cellules de  $\mathcal{T}_h$ .

Il s'agit d'un graphe orienté  $G = (V, A)$  tel que:

- $V = (V_1, \dots, V_n)$ ,  $n \leq d$ , où  $V_k$  désigne l'ensemble des  $k$ -cellules de  $M$ .
- $A = (A_1, \dots, A_n)$ , où  $A_k$  désigne les relations d'incidence entre paires de cellules de  $V_i \times V_j$ .

Un exemple de graphe d'incidence d'un maillage non structuré est donné à la figure 1.8. Une notion qui en est très proche est celle des diagrammes de HASSE qui servent à représenter les relations entre éléments d'ensembles partiellement ordonnés.

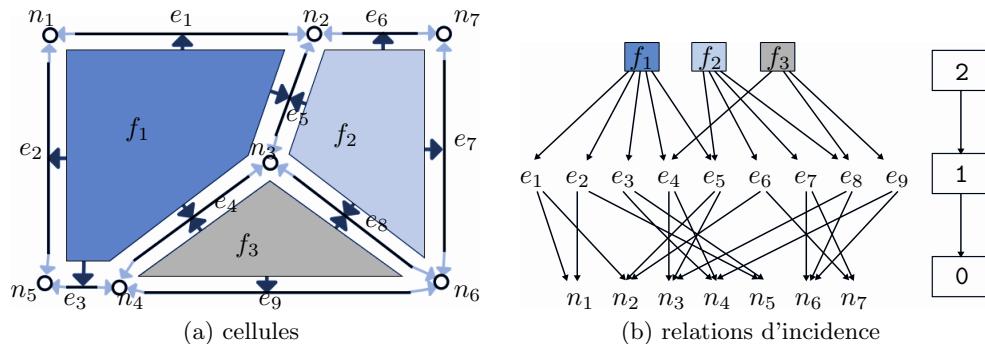


Figure 1.8: Exemple de topologie d'un maillage non-structuré [2].

<sup>10</sup>À proprement parler, la métrique  $g_p$  est une forme bilinéaire symétrique. La différence entre variété riemannienne et pseudo-riemannienne  $(\Gamma, g_p)_{p \in \Gamma}$  est qu'une métrique pseudo-riemannienne  $g_p$  n'est pas forcément définie positive.

En fait, on peut distinguer plusieurs modèles de représentation, selon le type et la richesse des connectivités que l'on veut disposer. L'idée est de ne pas stocker toutes les cellules et leurs relations mais seulement celles nécessaires à l'algorithme. En supposant qu'on stocke les points et les mailles et que chaque maille est définie par un triplet de points, les modèles les plus basiques sont :

- $0 \rightarrow 0$  : chaque point connaît ses points voisins. Sans donnée supplémentaire, il ne permet pas de retrouver directement le voisinage d'une maille.
- $0 \rightarrow 3$  : chaque point connaît ses mailles incidentes. Le voisinage d'une maille  $K$  s'obtient en intersectant les mailles incidentes à chaque sommet  $p_0, p_1, p_2$  de  $K$ ;
- $3 \rightarrow 3$  : chaque maille connaît ses mailles voisines. Le voisinage d'un point  $p$  s'obtient en énumérant les voisins  $K_j \in \mathcal{N}[K_i]$  tel que  $p \in K_i \cap K_j$ .

Un exemple est donné à la figure 1.9.

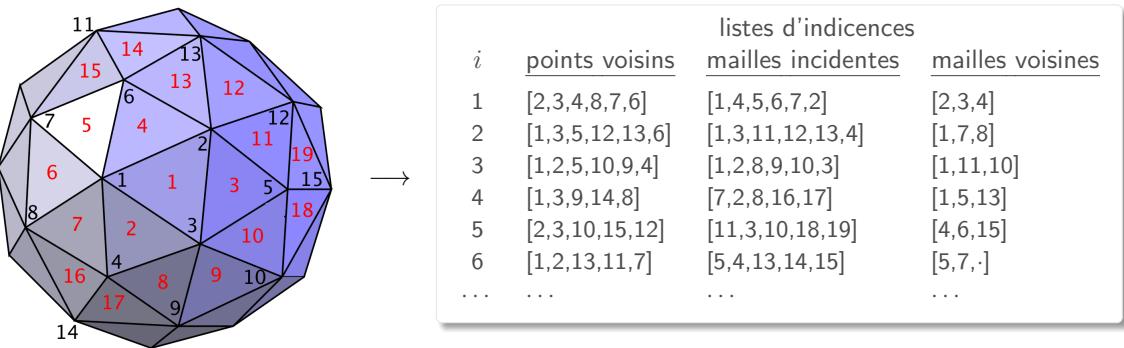


Figure 1.9: Représentation d'une surface triangulée par un graphe d'incidence.

MODÈLES IMPLICITES. Ici, la topologie est portée par un seul type d'élément sur lesquels les relations d'adjacence et d'incidence des cellules seront ensuite construites. L'avantage est que toutes les cellules de type et dimension différentes sont représentées *implicitement* et *identiquement*. Parmi les représentations les plus répandues, l'élément abstrait de base est l'**arête** (ou partie d'arête). Les modèles implicites usuels sont donnés à la figure 1.10 pour le cas des maillages surfaciques :

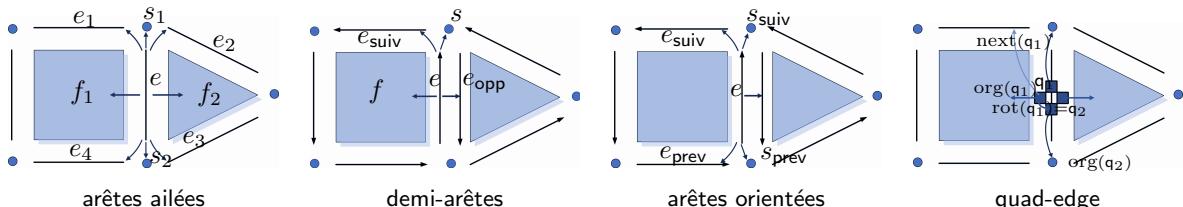


Figure 1.10: Représentation implicite de maillages surfaciques à base d'arêtes [2]. Ici, les points et les mailles sont présentes mais leurs connectivités sont portées exclusivement par les arêtes.

- **arêtes ailées** : ici chaque arête référence les noeuds et faces incidents, ainsi que les quatre arêtes partageant une face et un point avec elle.
- **demi-arêtes** : cette fois, une arête est dupliquée pour chaque face incidente. Chaque demi-arête référence la face qui la contient, le point vers lequel elle pointe, la demi-arête opposée et la demi-arête suivante dans la même face.
- **arêtes orientées** : dérivé de (b), chaque arête est également dupliquée. Une arête orientée appartient à une face, et référence uniquement les arêtes précédente et suivante, ainsi que les points précédents et suivants.
- **quad-edge** : plus général que les trois autres, il permet de représenter des variétés orientables ou non. En fait il permet de représenter le maillage primal et dual  $M^1, M^2$  d'une surface. Ici une

arête est décomposée en quatre brins, chacun pointant sur une face et un sommet incident au brin. Si un brin  $e$  pointe sur respectivement une face et un point dans  $M^1$ , alors  $M^2$  se construit en faisant pointer  $e$  sur respectivement un point et une face.

MODÈLES ORDONNÉS. L'inconvénient de ces représentations à base d'arêtes est qu'elles sont difficiles à généraliser. Cela est dû au fait que l'élément de base (arête) est dépendant de la dimension. Les modèles ordonnés comme les *cartes combinatoires* de dimension  $n$  au sens de la définition 15, et les *cartes généralisées* fournissent le degré d'abstraction nécessaire permettant de représenter des variétés orientables constituées de cellules quelconques en dimension quelconque.

**Définition 15** (CARTE COMBINATOIRE). *Une  $d$ -carte, ou carte combinatoire de dimension  $d \geq 0$ , est une algèbre  $C = (B, \beta_1, \dots, \beta_d)$  telle que :*

- $B$  est un ensemble fini d'éléments appelés brins;
- $\beta_1$  est une permutation sur  $B$ ;
- $\beta_i$  est une involution sur  $B$  avec  $2 \leq i \leq d$ ;
- $\beta_i \circ \beta_j$  sont des involutions sur  $B$ , avec  $1 \leq i < i+2 \leq j \leq d$ .

Dans une  $d$ -carte,

- $\beta_1$  permet de parcourir les brins qui composent une face commune ; elle relie les demi-arêtes d'une face entre elles.
- $\beta_2$  permet de passer d'un brin d'une face au brin d'une autre face ; elle relie les faces d'un plan ou d'une surface entre elles.
- $\beta_3$  permet de passer d'un brin appartenant à une maille volumique (hexaèdre, prisme, tétraèdre, pyramide etc.) au brin appartenant à une maille voisine. (voir figure 1.11)

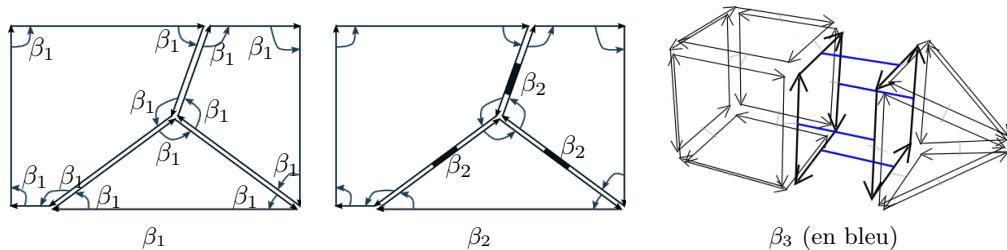


Figure 1.11: Relations d'incidence dans une carte combinatoire [2].

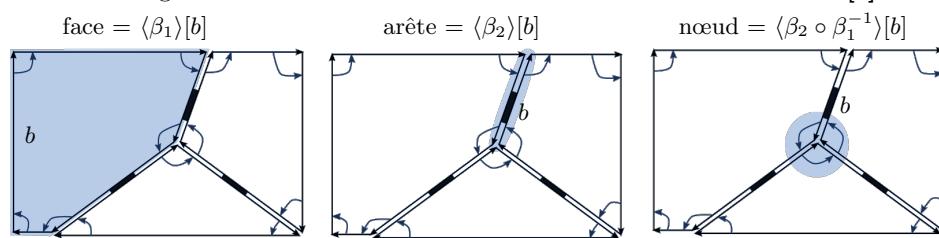


Figure 1.12: Reconstitution des cellules à partir d'un brin dans une 2-carte [2].

EN PRATIQUE. Ici, la topologie est entièrement portée par les brins. En calcul numérique, il est néanmoins utile d'accéder aux relations d'incidence des autres cellules (reconstruction de valeurs sur un point, transfert de flux à travers une arête etc.). Ainsi, la  $d$ -carte est souvent enrichie des autres cellules (points, faces et mailles). Ces dernières ainsi que leurs voisinages sont définies par composition judicieuse des  $\beta_i$  donnée à la définition 16. Leur reconstitution au sein d'une 2-carte est illustrée à la figure 1.12.

**Définition 16** (**k**-CELLULE). Soit  $C = (B, \beta_1, \dots, \beta_d)$  une **d**-carte.

La cellule de dimension  $k$  incidente à un brin  $b \in B$ , notée  $\varsigma_b^k$ , est définie par :

$$\varsigma_b^k = \begin{cases} \langle \beta_2 \circ \beta_1^{-1}, \dots, \beta_d \circ \beta_1^{-1} \rangle [b] & \text{si } k = 0 \\ \langle \beta_1, \dots, \beta_{i+1}, \beta_n \rangle [b] & \text{sinon} \end{cases}$$

EFFICACITÉ. En pratique, un modèle topologique est implanté par le biais d'une structure de données abstraite. Il s'agit d'une représentation du maillage muni de primitives de consultation et de modification des relations d'incidence entre chaque cellule. En notant  $n$  le nombre de points du maillage  $\mathcal{T}_h$ , on distingue usuellement trois ordres de grandeur pour le coût d'accès aux relations de voisinage :

- $O(1)$  si la relation d'incidence est directement stockée;
- $O(c)$  si elle peut-être retrouvée en  $c$  étapes,  $c \ll n$  étant la taille d'un voisinage local;
- $O(n)$  si elle nécessite potentiellement de parcourir tout le domaine pour la reconstituer.

Les modèles ordonnés tel que les **d**-cartes fournissent un accès à toutes les relations d'incidence en  $O(c)$ . Pour les graphes d'incidences, cela dépend des relations réellement stockées.

## 1.2 NOTIONS DE PARALLÉLISME

Dans la section précédente, nous avons présenté les types de surfaces gérées, ainsi que la représentation et le stockage des surfaces triangulées. Notons juste que ce dernier point est important pour les noyaux de remaillage en contexte massivement multithread. En effet la manière dont on représente la topologie impacte à la fois synchronisation des threads pour la consistance des données du maillage, mais aussi la localité des noyaux. Mais avant d'aborder ces aspects, nous décrivons les notions requises pour la mise en œuvre et l'évaluation d'un algorithme parallèle.

RAPPEL. Le calcul parallèle vise à accélérer le temps d'exécution d'un algorithme sur une instance de problème de taille fixe, ou à résoudre des instances de très grande taille sur une durée fixe.

### 1.2.1 Architectures de calcul

CLASSIFICATION. Bien qu'ils soient relativement génériques, les modèles de programmation parallèle sont liés aux architectures de calcul. En effet, la manière dont on parallélise un algorithme dépendra du type de parallélisme supporté par le hardware. Ainsi il est nécessaire de connaître sa topologie avant de réaliser le portage de l'algorithme. Dans ce cadre, la classification de Flynn, décrite à la table 1.1, permet de caractériser les architectures en fonction des flots de données et de contrôle.

Table 1.1: Classification de Flynn.

		instructions	
		single	multi
data	single	SISD	MISD
	multi	SIMD	MIMD

Dans cette taxonomie, SISD correspond aux machines séquentiels : une seule instruction est exécutée sur une seule donnée, à tout instant. D'un autre côté, une machine MISD (très rare) permet l'exécution de plusieurs instructions sur une même donnée, et est utilisé dans certains équipements critiques. Par ailleurs, une machine SIMD permet l'exécution d'une même instruction sur plusieurs données simultanément (par pipelining ou vectorisation). Enfin, MIMD correspond aux machines multi-processeurs : chaque processeur peut exécuter des instructions différentes sur des données différentes. Ces machines peuvent être classées selon leur topologie mémoire ainsi que leur espace d'adressage qui peuvent être (virtuellement) partagés ou distribués.

**MÉMOIRE.** Dans notre cas, nos architectures cibles correspondent aux machines MIMD à plusieurs cores et à mémoire partagée (multicore, manycore ou GPU). Ainsi l'espace d'adressage virtuel est commun à tous les cores, ce qui est adapté pour du multithreading. Selon le mode d'accès-mémoire supporté par ces machines, on distingue trois classes d'architectures : COMA (mémoire locale se comportant comme un cache, pas de replication de données en cas d'accès distants), UMA (uniforme), cc-NUMA (non uniforme avec protocole de cohérence de caches). Ici, la RAM peut être physiquement ou virtuellement partagée par le biais d'un DSM<sup>11</sup>. Les threads communiquent par le biais de variables globales ou segments de mémoire partagée, tandis que les processus peuvent communiquer directement via un protocole RDMA<sup>12</sup>. Dans les deux cas, la consistance des données doit être gérée explicitement. Ici, le coût des accès de données sont relatives à la latence mémoire qui peut être uniforme dans le cas des machines UMA, ou inégale dans le cas des machines NUMA.

**CACHES.** Afin de réduire le coût des accès-mémoire, les machines multicore ou manycore récents intègrent plusieurs niveaux de caches (deux ou trois en général). Ce sont des mémoires intermédiaires entre le CPU et la RAM, avec une capacité et une latence nettement réduites comparées à la RAM. À cette fin, un cache fournit de manière transparente les données qui ont été chargées dans un passé proche. Notons toutefois que les politiques de chargement et remplacement de blocs de données au sein d'un cache obéissent à un principe dit de localité au sens de la définition 17.

**Définition 17 (LOCALITÉ).** *Le principe de localité stipule que les données d'un programme ne sont pas accédées de manière statistiquement uniforme. Il y a plusieurs types de localités dont :*

- temporelle : les données récemment accédées seront très probablement bientôt réutilisées;
- spatiale : les données voisines seront très probablement bientôt accédées.

Afin de réduire le coût des accès-mémoire et obtenir un bon rendement d'utilisation des caches, il est donc nécessaire de structurer les instructions de l'algorithme de manière à maximiser la réutilisation de données et le référencement de données voisines. Cela peut être fait par la *renumérotation* de maillages ou de matrices creuses en calcul numérique, des techniques de *cache blocking*, ou encore le *prefetching*. Notons toutefois que n'est pas toujours possible si les accès-mémoire sont non prédictibles.

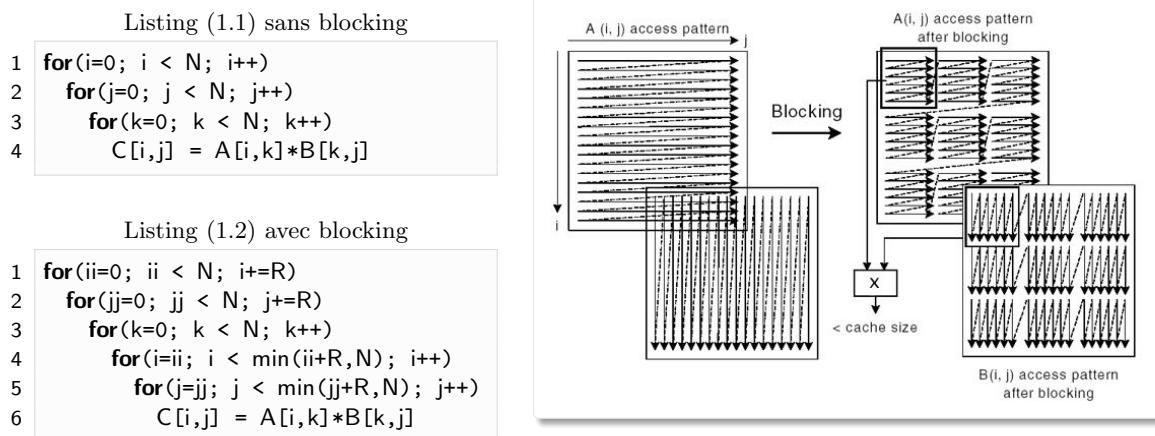


Figure 1.13: Multiplication matricielle efficace en cache [Intel].

**EXEMPLE.** Un cas concret de restructuration d'instructions par cache-blocking est donné à la figure 1.13 pour la multiplication de matrices denses (de taille N). Il consiste à découper les deux matrices carrées en  $m$  blocs de taille R, avec R la taille d'une ligne de cache de la machine cible. Pour chaque

<sup>11</sup>DSM pour *Distributed-Shared Memory* : couche logicielle permettant d'émuler un espace d'adressage global sur un machine à mémoire distribuée.

<sup>12</sup>RDMA pour *Remote Direct Memory Access* : protocole réseau permettant un accès direct des processus à une mémoire distante (zéro copie), sans intervention du CPU ou du noyau.

ligne d'une matrice, le fait d'itérer sur les  $N$  colonnes va induire un nombre important de défaut de cache. En effet celui-ci va systématiquement être purgé dans ce cas. Par contre, on va mieux réutiliser les coefficients de chaque matrice en organisant le calcul par bloc. Notons que cela est possible car on sait exactement quelle donnée on va accéder à un instant  $t$  de l'exécution : ici les motifs d'accès-mémoire sont prévisibles. Ce n'est malheureusement pas le cas pour nos noyaux de remaillage.

MULTITHREADING. Nos architectures cibles intègrent une forme de parallélisme très fin au niveau des instructions d'un core du processeur. Rappelons qu'une instruction est traitée en plusieurs phases (`fetch`, `decode`, `memaccess`, `execute`, `write` etc.) par le CPU. Au sein d'un core d'un CPU superscalaire<sup>13</sup>, on dispose d'un pipeline d'instructions qui permet de traiter chaque phase sur plusieurs instructions en même temps<sup>14</sup>. Dans ce type de core, un cycle<sup>15</sup> peut comporter des périodes d'inactivités (ou bulles) dans le pipeline, dues à des facteurs divers comme les dépendances d'instructions, une mauvaise prédiction de branchements, ou encore la latence due à un accès-mémoire. Une manière d'y remédier consiste à permettre plusieurs threads d'utiliser ces slots inoccupés. Notons néanmoins que remplir efficacement le pipeline est un problème réellement complexe car on doit gérer les dépendances d'instructions de plusieurs threads cette fois.

Selon la manière dont on remplit ces bulles, on dispose de plusieurs niveaux de parallélisme décrits à la figure 1.14. En (1) les unités fonctionnelles (ALU, FPU, etc.) sont réservées à un seul thread, tandis qu'en (2) elles sont disponibles à plusieurs threads mais sur des blocs de cycles CPU distincts. En (3), ces unités sont disponibles à plusieurs threads sur des cycles CPU différents (pas forcément des blocs), tandis qu'en (4) elles peuvent carrément être allouées à plusieurs threads sur le même cycle CPU.

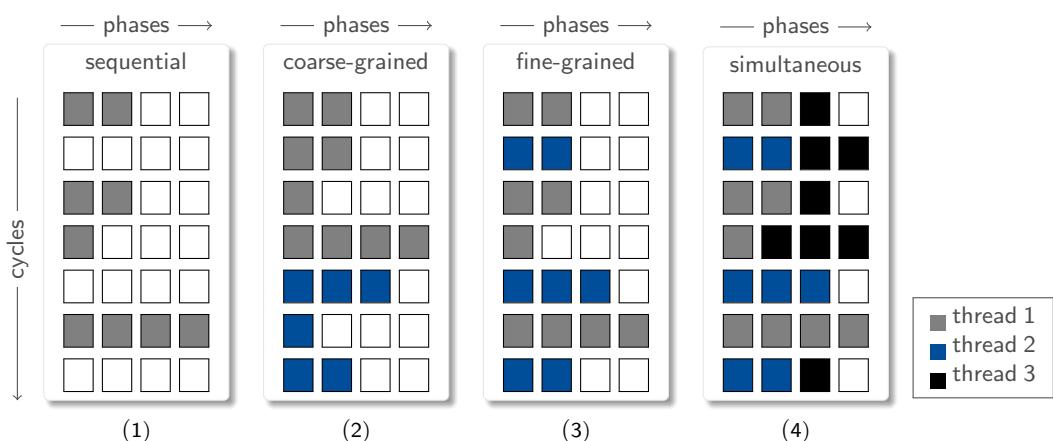


Figure 1.14: Pipeline d'instructions d'un core d'un processeur superscalaire et multithreading.

**Définition 18 (SIMULTANEOUS MULTITHREADING).** Il s'agit d'une forme de parallélisme au niveau des instructions d'un core d'un processeur superscalaire. Il désigne le fait de partager les ressources du core (unités de calcul et caches) entre plusieurs threads sur un même cycle CPU. Ainsi, il permet de réduire les bulles horizontales<sup>a</sup> et verticales<sup>b</sup> au sein du pipeline d'instructions.

<sup>a</sup>On a une bulle horizontale quand un slot dédié à une phase n'est pas occupé pour un cycle CPU donné.

<sup>b</sup>On a une bulle verticale dans le pipeline quand le CPU ne peut traiter aucune instruction dans un cycle entier

Le recours au SIMULTANEOUS MULTITHREADING (ou hyperthreading sur les puces Intel, cf. définition 18) permet de remplir les slots inoccupés du pipeline (dues à un accès-mémoire ou l'attente d'un résultat

<sup>13</sup>Il s'agit d'un core de processeur permettant l'exécution de plusieurs instructions simultanément grâce à un pipeline, et capable de détecter les dépendances d'instructions.

<sup>14</sup>Pour faire simple, on peut faire un `fetch` sur plusieurs instructions sur un même cycle CPU. Néanmoins, on ne peut pas faire un `fetch` et un `decode` simultanément sur une même instruction

<sup>15</sup>Un cycle correspond à la terminaison de toutes les phases de traitement d'une instruction par le CPU : `fetch`, `decode`, `memaccess`, `execute`, `write` etc.

d'une instruction précédente par exemple) en permettant à d'autres threads d'exploiter les unités fonctionnelles du core (ALU, FPU etc.). Ainsi il permet de masquer les pénalités liés à la latence, ce qui est très intéressant pour les processeurs manycore à large bande-passante mais à forte latence mémoire. Notons néanmoins que le gain de performances qu'il procure n'est pas toujours garanti. En effet, comme chaque thread doit charger ses propres données, cela peut entraîner une perte de localité en cache (voir définition 17), ainsi qu'une forte contention et/ou une saturation du cache partagé. Par conséquent, cela peut engendrer un nombre plus important de défauts de cache qu'en séquentiel.

### 1.2.2 Modèles de parallélisme

Nous avons vu le parallélisme d'instructions supportées par nos architectures cibles au paragraphe précédent. Cette fois, nous allons voir les deux formes de parallélisme plus haut niveau fréquemment utilisées en calcul haute performance.

DATA-PARALLEL. Ici, le parallélisme provient de la distribution de données entre les cores du CPU. En calcul numérique, il correspond au partitionnement du maillage en un ensemble de parties de tailles homogènes ou non, qui seront ensuite affectées aux processus ou threads comme illustré à la figure 1.15. Il correspond à un modèle d'exécution SPMD où chaque core exécute le même programme mais sur des données différentes. Sur l'exemple à la figure 1.15, les mailles grisées correspondent aux parties du domaine répliquées sur chaque core. Dans ce cas, chaque partie est traitée indépendamment, mais les processus doivent se synchroniser pour maintenir la cohérence des données aux interfaces. Cette forme de parallélisme se retrouve dans les algorithmes de calcul et visualisation haute performance. En mémoire partagée, elle peut-être implémentée à l'aide de directives de compilation tel qu'OpenMP ou de bibliothèques parallèles tel que Kokkos.

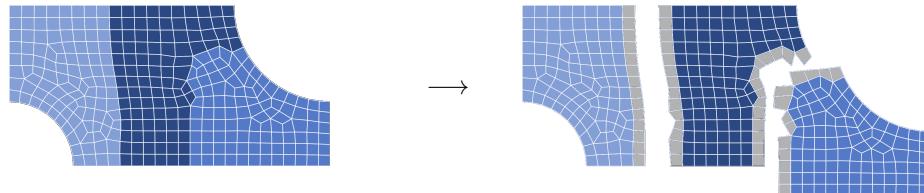


Figure 1.15: Partitionnement du maillage de calcul sur 3 cores. Ici les parties grisées correspondent aux mailles fantômes, c'est-à-dire des parties répliquées des interfaces du domaine.

TASK-PARALLEL. Cette fois, le parallélisme provient de la distribution du calcul entre les cores. Dans ce cas, l'algorithme est décomposé en tâches qui peuvent être scindées en sous-tâches, et qui vont être ordonnancées sur les cores. Ce genre d'algorithme se modélise naturellement à l'aide d'un graphe de calcul décrit à la définition 19. En raison des dépendances de tâches, des points de synchronisation ou d'échanges de données sont souvent nécessaires entre les sous-tâches. La parallélisation par tâches est usuellement utilisée pour les problèmes irréguliers incluant des boucles non statiquement bornées, les algorithmes récursifs et les schémas producteurs-consommateurs. En effet, elle permet naturellement de rééquilibrer la charge entre les cores au cours de l'exécution. Elle peut-être implémentée à l'aide de directives OpenMP (version 4), ou de bibliothèques parallèles tels que Cilk et TBB.

**Définition 19 (GRAPHE DE CALCUL).** *C'est une décomposition d'un algorithme en tâches en vue de leur ordonnancement. Il s'agit d'un graphe acyclique orienté et pondéré  $G = (V, A, \omega)$  où :*

- $V$  représente l'ensemble des tâches à ordonner,
- un arc  $e : (t_i, t_j) \in A$  représente une dépendance entre la tâche  $t_i$  et  $t_j$ ,
- et  $\omega : V \rightarrow \mathbb{N}$  correspond à l'affectation des coûts ou durées des tâches.

*Notons que si l'il existe un ordre partiel dans  $G$ , alors l'algorithme est parallélisable.*

Soit  $G = (V, A, \omega)$  l'exemple de graphe de calcul représenté à la figure 1.16. À nombre fixe  $p$  de cores, l'ordonnancement des tâches de  $G$  doit respecter les contraintes de précédence décrites à la

figure 1.16<sup>2</sup>. Ici, le temps de calcul  $T_p$  correspond au temps d'exécution de la tâche se terminant en dernier. La longueur du chemin critique<sup>16</sup>  $\ell_\infty = (c_i)_{i=1}^k$  du graphe donne une borne minimale sur  $T_p$  :

$$\forall p \in \mathbb{N}, |\ell_\infty| = \sum_{i=1}^k \omega[c_i] \leq T_p. \quad (1.6)$$

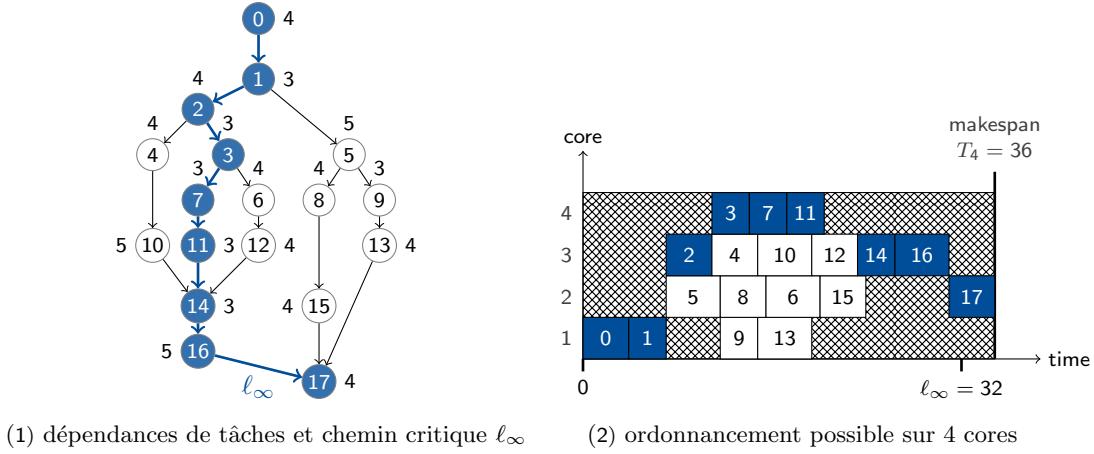


Figure 1.16: Exemple de graphe de calcul et ordonnancement possible sur 4 cores.

Dans ce cas, l'accélération théorique (voir définition 23) de l'algorithme est bornée par :

$$\lim_{p \rightarrow \infty} S_p = \frac{\sum_{t_i \in V} \omega[t_i]}{|\ell_\infty|}. \quad (1.7)$$

Ainsi, le gain par rapport à une exécution séquentielle n'excèdera pas cette borne qu'importe le nombre de cores utilisées.

**GRANULARITÉ.** En fait pour un algorithme et un modèle de parallélisme donné, on peut extraire plusieurs niveaux de parallélisme selon la granularité désirée, au sens de la définition 20.

**Définition 20 (GRANULARITÉ).** Elle représente la quantité de calcul utile local entre deux points de synchronisation d'un algorithme parallèle donné.  
Elle désigne également la finesse de décomposition en tâches de cet algorithme.

Ainsi la granularité détermine le nombre de tâches ainsi que le nombre d'instructions par tâche que l'on attribue à chaque core pour un algorithme donné :

- si elle est *fine* alors les périodes de calcul sont relativement courtes par rapport à celles des communications. Ainsi, on a une meilleure répartition de charges des cores, mais on peut avoir un surcoût important due à la synchronisation.
- si elle est *grossière* alors on a un faible ratio de temps de communications sur temps de calcul. Ainsi on passe plus de temps à faire du calcul, néanmoins on peut être confronté à un déséquilibre de charges important.

In fine, le choix de la granularité dépend de la structure de l'algorithme (nombre d'instructions par tâche, patterns de communication des processus), mais également de la machine cible (nombre de cores, fréquence et mémoire par core, latence et bande-passante mémoire).

<sup>16</sup>Le chemin critique d'un graphe de calcul est le plus long chemin du sommet source vers un sommet puits. Sa longueur correspond au temps de restitution minimal de l'algorithme même quand il s'exécute sur une infinité de cores.

IRRÉGULARITÉ. Nous avons vu dans l'introduction que nos noyaux de remaillage étaient des algorithmes irréguliers : les dépendances de tâches évoluent dynamiquement et de manière non prévisible d'une part, et les instructions sont dominées par des accès-mémoire avec une faible localité.

**Définition 21** (PARALLÉLISME AMORPHE). *Il s'agit de la forme de parallélisme inhérent aux algorithmes irréguliers [164]. En raison des dépendances de données non prévisibles, il implique :*

- *des tâches indépendantes qui peuvent devenir conflictuelles,*
- *des tâches générées de manière non déterministe au cours de l'exécution;*
- *des motifs irréguliers d'insertion ou de suppression de données.*

Ainsi si on construit le graphe de calcul d'un noyau donné, alors ce graphe changerait tout le temps car les dépendances évoluent en fonction des données. Ainsi le parallélisme inhérent à ce type de problème n'a pas de structure claire, il est qualifié d'amorphe au sens de la définition 21.

ASYNCRONISME. Le problème majeur induit par le parallélisme amorphe est qu'il peut induire un important déséquilibre de charges. En effet même si on attribue initialement le même nombre de tâches aux cores, ils ne mettront pas nécessairement le même temps pour traiter ces tâches puisqu'elles vont chacune engendrer d'autres tâches de manière non prévisible. Une manière d'y remédier consiste à recourir à une *exécution asynchrone* au sens de la définition 22.

**Définition 22** (ASYNCRONISME). *Soit  $G = (V, E, \omega)$  un graphe de calcul. Notons  $[d_i, f_i]$  les dates de début et de fin de chaque tâche  $t_i \in V$  et  $\prec$  une relation de précédence causale des tâches. On dit qu'une séquence de tâches  $(t_0, \dots, t_n)$  admet une exécution asynchrone si :*

$$\forall i, j \in [0, n] : i < j \Rightarrow d_i \prec d_j \wedge f_i \not\prec d_j, \quad (1.8)$$

*Autrement dit, l'asynchronisme désigne le fait que  $t_i$  peut débuter avant la fin de  $t_{i-1}$  pour une séquence de tâches  $(t_0, \dots, t_n)$ . Ainsi un algorithme est dit asynchrone si les communications sont non bloquantes et recouvertes par du calcul local dans un paradigme à passage de messages.*

Ainsi l'asynchronisme permet de réduire le surcoût lié aux synchronisations au sein d'un algorithme parallèle par recouvrement des communications. En particulier, il permet de masquer les pénalités liées à la latence lors d'un accès-mémoire distant ou d'un échange de messages. Néanmoins, il est complexe à exhiber en raison de la consistance des données et la convergence qui ne sont plus garanties.

### 1.2.3 Évaluation pratique

Pour évaluer les performances d'un algorithme parallèle, il faut identifier les sources d'inefficacité. Pour cela, on distingue trois états possibles des cores, et on mesure les temps passés dans chacun de ces états, c'est-à-dire le temps de :

- **calcul** ( $t_{\text{calc}}$ ) : nombre de cycles passés à faire du calcul utile.
- **communication** ( $t_{\text{comm}}$ ) : nombre de cycles passés à réaliser les communications ou synchronisations entre les cores. Il dépend du volume de données, du débit et latence du médium<sup>17</sup>.
- **attente** ( $t_{\text{wait}}$ ) : nombre de cycles perdus à attendre un événement ou une synchronisation. Il s'agit le temps d'inactivité du core. Il est caractéristique d'un déséquilibre de charges entre les cores, ou d'un mauvais entrelacement des calculs et communications.

Ainsi le temps de restitution de l'algorithme sur  $p$  cores est donné par :

$$T_p = \max_{i=1}^p \left[ t_{\text{calc}}^{[i]} + t_{\text{comm}}^{[i]} + t_{\text{wait}}^{[i]} \right]. \quad (1.9)$$

SCALING. En fait, évaluer l'algorithme implique de mesurer le gain obtenu par rapport à sa version séquentielle, ou bien le rendement d'utilisation des cores. De manière concrète, cela s'exprime par l'accélération et l'efficacité, au sens des définitions 23 et 24.

<sup>17</sup>Les communications peuvent se faire par accès-mémoire distant direct (RDMA) ou par passage de messages

**Définition 23** (ACCÉLÉRATION). *Elle mesure le gain sur le temps de restitution d'un algorithme parallèle A par rapport au meilleur algorithme séquentiel A\*, ou à défaut à la version séquentielle de A. En notant  $T_p$  le temps de restitution sur p cores, elle est donnée par :  $S_p = \frac{T_1}{T_p}$ .*

**Définition 24** (EFFICACITÉ). *Elle mesure le rendement par core, et est donnée par :  $E_p = \frac{T_1}{p T_p}$ .*

In fine, ce qui est réellement intéressant c'est de voir comment ce gain évolue quand on fait varier le nombre de cores actifs, ou bien de la taille de l'instance du problème. Autrement dit, cela consiste à évaluer sa scalabilité (ou passage à l'échelle) au sens de la définition 25.

**Définition 25** (SCALABILITÉ). *Elle mesure la capacité de l'algorithme à maintenir l'efficacité en cas de montée en charge, soit en termes de données, soit en unités de calcul.*

*Soit p le nombre d'unités de calcul, et n la taille du problème considéré. Dans ce cas,*

- la scalabilité forte mesure l'évolution de  $E_p$  en fonction de p pour n fixe,
- la scalabilité faible mesure l'évolution de  $E_p$  quand p et n évoluent linéairement.

En fait, l'accélération n'est pas nécessairement proportionnelle au nombre de cores utilisés, contrairement à l'intuition. En effet, elle est déjà limitée par le surcoût induit par la parallélisation. Mais de plus, elle est sévèrement limitée par la partie séquentielle, aussi infime soit elle (voir théorème 1).

**Théorème 1** (LOI D'AMDAHL). *Elle donne une borne sur la scalabilité forte d'un algorithme.*

*Soient*

- p le nombre de cores utilisés,
- $\ell_1$  et  $\ell_p$  les durées des parties séquentielle et parallèle,
- $\ell_\infty = \ell_1 + \ell_p$  le temps de restitution théorique,
- et enfin  $\alpha = \frac{\ell_1}{\ell_\infty}$  le ratio de durée de la partie séquentielle sur le temps total théorique.

*Alors le temps de restitution réel est :  $T_p = \alpha T_1 + (1 - \alpha) \frac{T_1}{p}$ .*

*Dans ce cas, la scalabilité forte vaut :*

$$\lim_{p \rightarrow \infty} S_p = \lim_{p \rightarrow \infty} \frac{T_1}{T_p} = \lim_{p \rightarrow \infty} \frac{1}{\alpha + \frac{1-\alpha}{p}} = \frac{1}{\alpha}. \quad (1.10)$$

Ainsi d'après la loi d'Amdahl, si la partie intrinsèquement séquentielle vaut  $\alpha = 10\%$  alors l'accélération  $S_p$  ne peut excéder 10 qu'importe le nombre de cores utilisés. Par conséquent, l'algorithme doit être intégralement parallélisé pour espérer maintenir une accélération substantielle à nombre de cores élevé. Notons que ce n'est pas toujours évident en pratique.

\* \* \*

## Adaptation de surfaces triangulées.

Dans ce chapitre, nous donnons une brève synthèse des algorithmes impliqués dans le remaillage de surface. En premier lieu, nous montrons comment le maillage initial est générée selon la représentation fournie en entrée (nuage de points ou fonction de distance signée). Ensuite nous montrons en quoi le maillage obtenu la qualité du maillage impacte l'efficacité des solveurs en calcul numérique. Ces deux aspects nous permettra de mieux comprendre la nécessité et l'intérêt d'adapter le maillage. Enfin, nous donnons un panorama des techniques d'adaptation de maillages de surface utilisées dans la littérature.

HYPOTHÈSES. Dans notre cas, nous supposons que la surface fournie en entrée est une variété lisse par morceaux, orientable et sans bord pour les raisons que nous avons évoqués au chapitre précédent. Par ailleurs, nous supposons aussi qu'un maillage de surface est interpolant, au sens de la définition 5. Ainsi, on peut retrouver localement la surface au voisinage d'un point ou d'une maille par interpolation. Dans ce cadre, remailler une surface consiste à reconstruire une maillage interpolant de cette surface, afin de réduire l'erreur d'une solution numérique ou bien l'erreur de la surface elle même, tout en améliorant la qualité des mailles.

---

2.1	Extraction de la surface . . . . .	35
2.1.1	Cas d'un volume discret . . . . .	36
2.1.1.1	Discrétisation du volume . . . . .	37
2.1.1.2	Extraction de la surface . . . . .	38
2.1.2	Cas d'un volume implicite . . . . .	39
2.1.2.1	Extraction de la surface . . . . .	39
2.1.2.2	Résolution des ambiguïtés . . . . .	40
2.1.2.3	Stratégies accélératrices . . . . .	40
2.2	Adaptation de la surface . . . . .	42
2.2.1	Anisotropie et qualité d'un maillage . . . . .	42
2.2.1.1	Impact sur les solveurs numériques . . . . .	42
2.2.1.2	Forme et alignement optimal . . . . .	43
2.2.2	Approches basées sur une paramétrisation . . . . .	45
2.2.3	Approches variationnelles . . . . .	45
2.2.3.1	Répartition des points . . . . .	46
2.2.3.2	Noyaux globaux . . . . .	47
2.2.4	Approches locales . . . . .	48
2.2.4.1	Noyaux locaux . . . . .	48
2.2.4.2	Reconstruction de la surface . . . . .	49
2.2.4.3	Cartes de tailles . . . . .	50
2.3	Synthèse . . . . .	51

---

### 2.1 EXTRACTION DE LA SURFACE.

En fait pour un domaine fermé de  $\mathbb{R}^3$ , les maillages surfaciques et volumiques correspondants sont souvent intimement liées comme illustré sur la figure 2.1. En effet si le bord du domaine est décrit par un nuage de points, alors le maillage surfacique peut s'obtenir par extrudant le maillage volumique. Inversement, si l'on dispose déjà d'un maillage surfacique, alors le maillage volumique s'obtient en triangulant les points de cette surface, puis en raffinant le maillage obtenu. Par contre, le maillage à adapter diffère sensiblement selon la manière dont il a été généré, et plus précisément

de la représentation de la frontière du domaine considéré. En effet cette frontière peut être décrite explicitement par un nuage de points (ou encore une B-Rep), ou implicitement par le biais d'une fonction de distance signée décrivant l'intérieur et l'extérieur du domaine.

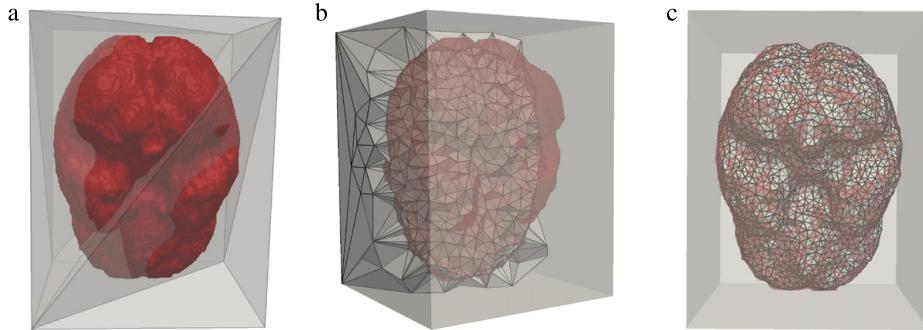


Figure 2.1: Reconstruction du maillage surfacique à partir du volumique [118].

Ainsi nous rappelons deux principales méthodes de triangulation volumique impliqués dans la génération du maillage initial pour ces deux types de représentations à la section 2.1.

### 2.1.1 Cas d'un domaine discret : méthodes de Delaunay

Elle regroupe les algorithmes de triangulation d'un nuage de points d'un domaine  $\Omega$  basés sur un critère dit de DELAUNAY décrit à la définition 26. En raison de sa robustesse et de ses propriétés particulières, il s'agit de l'approche de triangulation la plus répandue dans les applications numériques. Partant d'un nuage de points décrivant la frontière d'un domaine<sup>1</sup>, elle vise à discréteriser à la fois l'intérieur et la frontière de ce domaine par insertion successive de points, de sorte que la triangulation finale respecte ce critère de DELAUNAY.

**Définition 26** (CRITÈRE DE DELAUNAY). Soit  $S$  un ensemble fini de points de  $\mathbb{R}^3$ .

Une triangulation  $T_h$  de  $S$  correspond au recouvrement de l'enveloppe convexe de  $S$  par des simplexes<sup>a</sup>. On dit que  $T_h$  respecte le critère de Delaunay si l'intérieur de la boule circonscrite de tout simplexe de  $T_h$  ne contient aucun autre point de  $S$ .

Dans ce cas, on dit simplement que c'est une triangulation de Delaunay.

<sup>a</sup>Un simplexe est un triangle ou un tétraèdre.

OPTIMALITÉ. Tout nuage de points  $S$  admet au moins une triangulation de DELAUNAY. De plus celle-ci est unique dans la plupart des configurations<sup>2</sup>. En fait, elle est considérée comme la "meilleure" triangulation possible de  $S$  pour de nombreuses applications numériques en raison des propriétés décrites à la proposition 1.

**Propriété 1** (OPTIMALITÉ D'UNE TRIANGULATION DE DELAUNAY). Soit  $S$  un nuage de points d'un domaine planaire ou volumique  $\Omega$ , et  $T_h$  une triangulation de Delaunay de  $S$ .

Parmi toutes les triangulations possibles de  $S$ ,

- $T_h$  maximise l'angle minimum  $\theta_{\min}$  de chaque maille [4].
- $T_h$  est idéale pour l'interpolation de fonctions harmoniques  $u$  telle que  $\Delta u = 0$  [14, 15, 5].
- $T_h$  minimise l'erreur d'interpolation en norme  $L^p$  d'une forme quadratique  $u$  sur  $\Omega$  [16].

En fait, les deux premières propriétés sont les plus pertinentes. Dans le cas planaire, la première garantit un ratio d'aspect maximal des mailles sans relocalisation de points. La seconde garantit que  $T_h$

<sup>1</sup>On peut éventuellement l'obtenir à partir d'un échantillonnage d'une B-Rep

<sup>2</sup>Sauf dans le cas où il y a  $k \geq d + 2$  points co-sphériques de  $T_h$ , et le cas échéant il est toujours possible de se ramener d'une instance de Delaunay à une autre par le biais de transformations simples [3]

fournit la discréétisation la plus lisse possible d'un domaine  $\Omega$  pour l'interpolation d'un champ scalaire sur  $\Omega$ . Notons toutefois que ces propriétés ne sont plus forcément vérifiées dans le cas volumique, du fait que le critère de Delaunay ne garantit pas l'absence de tétraèdres aplatis appelées *slivers*.

### 2.1.1.1 Discréétisation du volume

Partant d'un nuage de points  $S$  décrivant la frontière du domaine  $\Omega$ , la première étape dans la construction d'une triangulation de Delaunay de  $\Omega$  consiste à insérer itérativement les points par le biais du *noyau de Delaunay*. Il est décrit par :

$$\mathcal{T}_h = \mathcal{T}_{h-1} - C_i + B_i \quad (2.1)$$

où  $C_i$  correspond à la cavité du point  $p_i$  constitué des mailles de  $\mathcal{T}_{h-1}$  dont la sphère circonscrite contient  $p_i$ , et  $B_i$  correspond au voisinage de  $p_i$  après ré-étoilement. Il s'implémente naturellement par l'algorithme de Bowyer-Watson [17, 18]. En pratique, le nuage de points est enrichi des sommets d'une boîte englobante de  $\Omega$  avant d'être triangulé comme illustré sur la figure 2.1. Ainsi cela assure qu'il existe toujours une maille de la triangulation courante contenant le point à insérer.

**BOWYER-WATSON.** A l'itération  $i$ , on commence par localiser la maille qui contient le point  $p_i$  par le biais d'un parcours barycentrique. Partant d'une maille source, on parcourt  $\mathcal{T}_h$  en évaluant le signe des coordonnées barycentriques de  $p_i$  dans  $K \in \mathcal{T}_h$ . Il y a alors trois cas de figures :

- soit ils sont tous positifs alors  $K$  contient  $p_i$ ,
- soit l'un d'eux est négatif alors il indique la maille voisine  $K_j$  à évaluer,
- soit deux d'entre eux sont négatifs alors il y a deux directions possibles qu'il faudra discriminer arbitrairement ou géométriquement (voir figure 2.2).

Il faut toutefois choisir la source de manière judicieuse dans le cas où le domaine comporte des trous ou une frontière concave comme sur la figure 2.2. Une fois la maille  $K$  identifiée, on procède à l'expansion de la cavité de  $p_i$ . Pour cela, on identifie les mailles de  $\mathcal{T}_{h-1}$  dont les sphères circonscrites contiennent  $p_i$ , grâce à un parcours en largeur de  $\mathcal{T}_{h-1}$  à partir de  $K$ . La dernière étape consiste ensuite à supprimer chaque  $K \in C_i$  de  $\mathcal{T}_{h-1}$ , et à former les mailles  $K_j \in B_i$  en reliant les faces opposées de chaque  $K \in C_i$  à  $p_i$ .

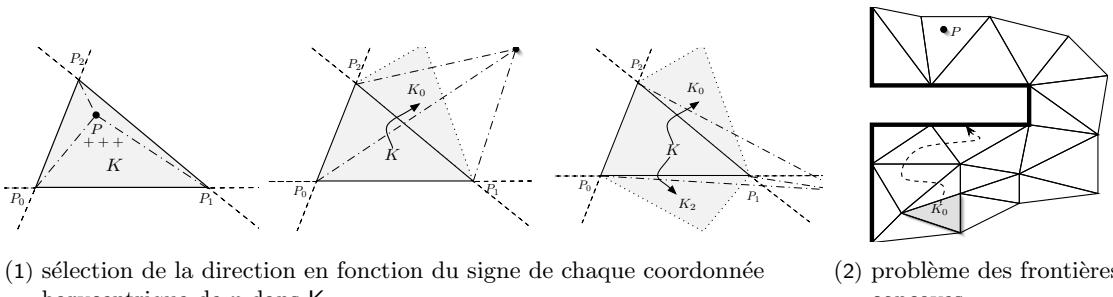


Figure 2.2: Localisation de maille par parcours barycentrique [60].

À noter qu'en pratique la construction de la cavité de chaque point soulève des problèmes numériques. Bien qu'il est théoriquement garanti que la cavité d'un point soit ré-étoilable, en pratique cela n'est pas forcément le cas en raison des erreurs d'arrondis en arithmétique flottante. Pour palier ce problème, on peut recourir à une procédure de correction de cavité comme dans [61] et/ou utiliser des prédicts géométriques basée sur une arithmétique flottante à précision adaptative tels que décrits dans [19].

**ALTERNATIVES.** Pour finir, il existe deux autres méthodes de triangulation de Delaunay : celles de Lawson [6] et la projection paraboloidale [20, 21]. La première est restreinte au cas planaire et est basée sur le fait qu'il est possible de rendre Delaunay toute triangulation planaire en utilisant uniquement aux bascules d'arêtes. La seconde exploite le fait qu'une triangulation de Delaunay d'un

ensemble de points  $S$  peut être vu comme une projection sur  $\mathbb{R}^d$  des enveloppes convexes inférieures des projetés de chaque  $p \in S$  sur une paraboloïde comme illustré sur la figure 2.3. Néanmoins ce lien entre triangulation de Delaunay et enveloppe convexe de points d'un paraboloïde n'est réalité utilisée que pour des preuves théoriques, et peu d'algorithmes exploitent cette propriété, car elle est coûteuse en pratique (voir [22] pour un exemple).

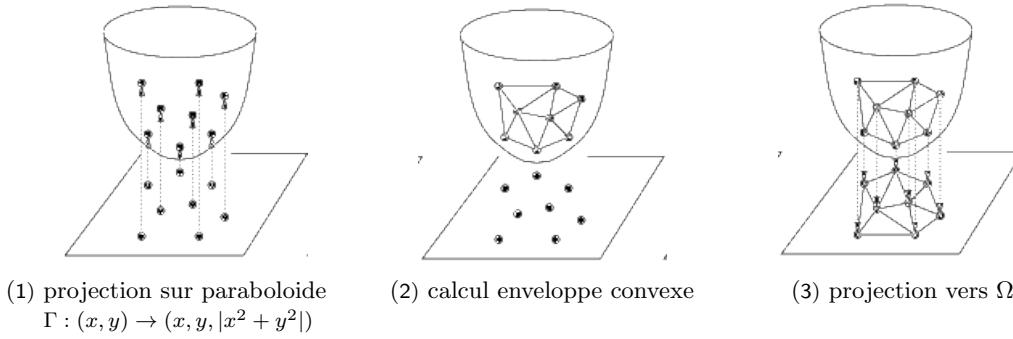


Figure 2.3: Extraction d'une triangulation de Delaunay par projection de points [7].

Le principe précédent est illustré à la figure 2.3 dans le cas planaire. En premier lieu, le nuage de points est projeté sur un paraboloïde formant un ensemble de points  $P$ . Ensuite on calcule l'enveloppe convexe  $C(P)$  de  $P$ . Enfin on projette les éléments de  $C(P)$  sur le domaine-plan. Ici, le gros du calcul porte sur l'extraction de  $C(P)$ . Néanmoins une vaste littérature est dédiée à ce sujet (cf. [23]).

### 2.1.1.2 Extraction de la surface

À l'issue de l'étape précédente, nous avons un maillage de la boîte englobante du domaine à discréteriser, tel que ses points internes soient sur la frontière de ce domaine (qui est la surface que nous voulons extraire). Ainsi nous avons déjà les points de la surface. Néanmoins il n'est pas garanti que les autres entités (arêtes ou faces) de la surface soient dans ce maillage, à moins que le domaine ne soit convexe. En fait, pour obtenir un maillage de surface, il faudrait appliquer un traitement spécifique sur le maillage précédent de manière à forcer la présence de ces entités.

RENFORCEMENT. Il y a plusieurs stratégies pour forcer la présence des entités de la frontière. Elles sont toutes basées sur le fait de rajouter des points additionnels (ou points de STEINER) pour renforcer le critère de Delaunay près de la frontière. On distingue deux écoles à ce sujet :

- DELAUNAY-CONFORME : ici, les entités de la frontière sont intégrés au maillage en enrichissant le nuage de points initial  $P$  par un ensemble fini de points de STEINER  $S$ , de manière à obtenir une triangulation de Delaunay de  $P \cup S$ . Ici,  $S$  peut être déterminé *a priori* [24], néanmoins il en faut souvent beaucoup pour satisfaire cette contrainte de conformité. En fait, cela a pour effet de sur-discréteriser le bord de  $\Omega$ , ce qui dégrade potentiellement la qualité des mailles. Bien que des approches plus pratiques existent [24, 25], la conformité est une contrainte trop forte en raison du nombre excessif de points de STEINER induit. Cela a motivé le développement des approches DELAUNAY-CONTRAINTE telles que [224, 26].
- DELAUNAY-CONTRAINTE : ici, les entités de la frontière sont insérées dans la triangulation de manière à ce que le critère de Delaunay soit respecté sur  $\Omega$ , sauf sur les entités **contraintes**. Dans ce cas, on a une triangulation de Delaunay dite **contrainte**. Il s'agit d'une triangulation  $\mathcal{T}_h$  dans laquelle la sphère circonscrite de chaque simplexe  $K \in \mathcal{T}_h$  ne contient aucun autre point de  $\mathcal{T}_h$  qui soit "visible" de l'intérieur de  $K$ . Initialement développée dans [27, 8], cette notion de "visibilité" peut être vue comme une relaxation de la contrainte de conformité. En pratique, cela implique un nombre beaucoup moins important de points de Steiner [4].

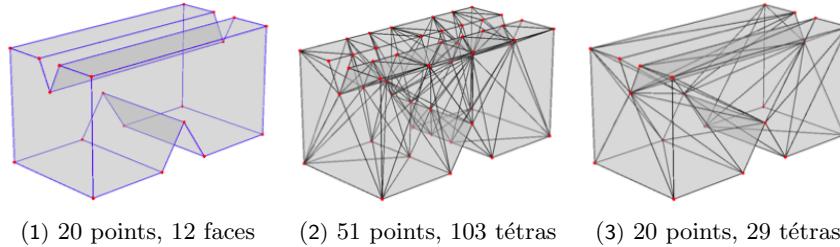


Figure 2.4: Renforcement conforme et contraint de la frontière d'un domaine [28].

Un comparatif des triangulations possibles d'un domaine polyédrique selon la stratégie appliquée pour le renforcement des entités de sa frontière est donnée à la figure 2.4 et repris de [28]. La représentation de la frontière par une B-Rep est donnée en (1). Les triangulations DELAUNAY-conforme et contrainte sont respectivement données en (2) et (3). Ici on voit que la stratégie **contrainte** minimise clairement le nombre de points de STEINER (aucun en l'occurrence), par contre on a des mailles très étirées sur la surface. En pratique, des versions robustes de ces algorithmes sont implémentés dans plusieurs logiciels open-source, notamment `triangle` [29], `tetgen` [30], `netgen` ou encore `CGAL` [9].

### 2.1.2 Cas d'un domaine implicite : marching cubes

Dans certaines applications, le domaine n'est ni paramétré ni spécifié explicitement par sa frontière. En visualisation médicale par exemple, ils sont souvent caractérisés comme des régions où les grandeurs mesurables sont définis par un seuil physiologique connu. Par exemple, les techniques de tomodensitométrie mesurent un nombre de relaxation de l'intensité des rayons X répandus dans le corps humain; Puisque l'air, l'os, l'eau ont des comportements différents par rapport à ce nombre, ces entités peuvent être observées séparément en regardant les ensembles de niveaux de la relaxation. En simulation numérique, les problèmes d'interfaces physiques libres et mobiles sont souvent traités par la méthode des ensembles de niveaux, ce qui inclut le maillage de géométries implicites [31].

**PRINCIPE.** Ici la surface est définie par les points correspondant aux zéros d'une équation implicite  $\phi$ . Ainsi elle représente la frontière d'un domaine défini par  $\phi(x) \geq 0$ : ici retrouver la surface revient à extraire les isosurfaces de  $\phi$ . De manière synthétique, elle repose sur deux étapes : l'échantillonnage de  $\phi$  en vue de créer un nuage de points, et la triangulation explicite de ces points [41]. Pour l'échantillonnage, une décomposition du domaine est faite par le biais d'une grille ou d'un octree, et la surface idéale est approchée localement par une surface plane par morceaux. Dans ce cadre, l'approche la plus intuitive et la plus utilisée est celle des *marching cubes*, à partir de laquelle de nombreuses extensions ont été développées. Une étude comparative de ces variantes est décrite dans [42].

#### 2.1.2.1 Extraction de la surface

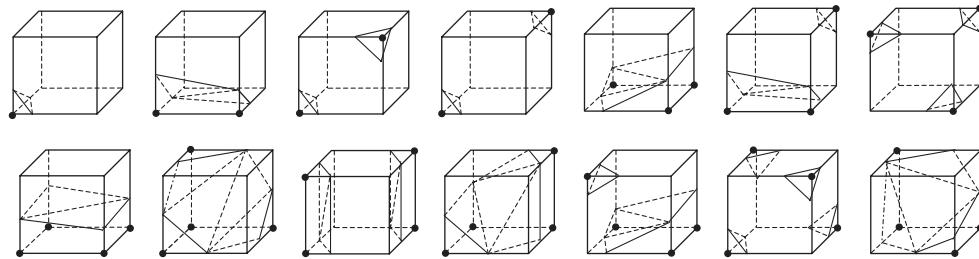


Figure 2.5: Intersections possibles de la surface avec chaque voxel selon le signe de  $\phi$  à ses sommets [43].

**IDÉE.** Initialement introduite dans [44], l'idée derrière les *marching cubes* est intuitivement simple. La surface est reconstruite par intersection des arêtes d'une grille de voxels avec le contour du domaine,

ce qui engendre la création de points. Elle s'obtient en utilisant différents motifs de polygonisation selon la manière dont chaque arête est intersectée au sein d'un voxel, comme sur la figure 2.5.

Plus précisément, la première étape consiste à construire une boîte englobante  $B$  du domaine à reconstruire. Ensuite, cette boîte  $B$  est discrétisée uniformément par une grille de  $n_x n_y n_z$  voxels<sup>3</sup>, et le calcul d'intersections se fait de manière itérative sur les voxels. Pour chaque voxel  $v_{ijk} \in B$ , on compte le nombre  $n_{ijk}^-$  de sommets tel que  $\phi(x) \leq 0$ . Ensuite on choisit une configuration dans une table de patterns qui recense les topologies possibles de triangles selon la valeur de  $n_{ijk}^-$ . Notons juste qu'il y a 2<sup>8</sup> configurations possibles au sein de chaque voxel. Néanmoins, un recours astucieux de symétries et de rotations permet de réduire ce nombre à 14 comme sur la figure 2.5. Sur cette figure, chaque point correspond aux sommets du voxel pour lesquels  $\phi$  est négatif. Enfin, un parcours géométrique des voxels permet la réutilisation des données interpolées sur les arêtes communes à plusieurs voxels.

### 2.1.2.2 Résolution des ambiguïtés

L'étape précédente ne garantit pas d'obtenir directement une reconstruction valide de la surface  $\Gamma$ . En effet il existe des configurations où la portion de surface n'est pas déterminée de manière unique par l'intersection de  $\Gamma$  avec les arêtes d'un voxel tel qu'illustré à la figure 2.6.

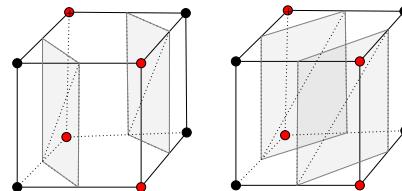


Figure 2.6: Configuration ambiguë induite par les *marching cubes*.

Une liste exhaustive de ces configurations ambiguës est donnée dans [45]. En fait, sans traitement particulier, le maillage résultant pourra contenir des trous ou autres inconsistances topologiques. Heureusement, il y a plusieurs stratégies pour résoudre ces ambiguïtés :

- *contours*: pour chaque voxel, une solution possible consiste à approcher localement la surface par un patch (selon les valeurs ponctuelles de  $\phi$ ) puis d'inférer le bon motif de discréétisation selon la configuration de ce patch [45].
- *ordre total* : les configurations ambiguës peuvent être discriminées par un ordre total imposé sur les points d'intersection de l'isosurface avec les arêtes du voxel comme dans [46]. Ainsi, la reconstruction discrète de la surface s'obtient par le graphe formé par ces points.
- *marching tetrahedra*: ici, la surface est capturée par découpage de voxel [47, 10, 48]. Le voxel peut être scindé en 6, 8 ou 12 tétraèdres<sup>4</sup>, et la surface est interpolée sur chaque tétraèdre. Notons qu'ici on a directement une reconstruction non ambiguë de la surface sur chaque voxel, ainsi que la possibilité d'adapter la discréétisation [49], contrairement à la version de base.

### 2.1.2.3 Stratégies accélératrices

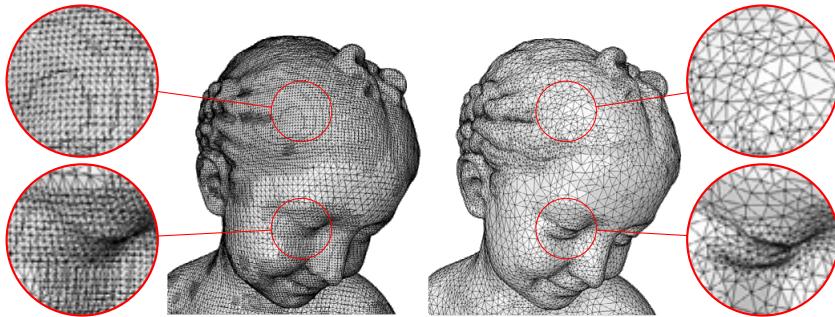
EXTENSIONS. En raison de leur compacité, les *marching cubes* sont intensivement utilisées pour le rendu temps réel et interactif de données volumétriques. Néanmoins cette contrainte temps-réel soulève une problématique relative à l'accélération de l'extraction des isosurfaces. À ce titre, bon nombre d'extensions ont été développées afin d'éviter les traitements sur les voxels inactifs ou vides, car près de 30 à 70% du temps de calcul impliquent ces voxels [50, 51] (voir [52–54] pour plus de détails sur ces variantes).

<sup>3</sup>Chaque voxel est identifié de manière unique par  $\{v_{ijk}\}_{i \in [1, n_x], j \in [1, n_y], k \in [1, n_z]}$

<sup>4</sup>Pour  $n = 8$ , il est scindé en quatre pyramides dont les bases correspondent aux faces du voxel, et chaque pyramide est ensuite scindé en deux tétraèdres. Pour  $n = 6$  ou 12, un point de contrôle fictif est créé au centre du voxel, et  $\phi$  est ensuite interpolée sur ce point.

Par ailleurs, ils présentent un fort potentiel de parallélisation, de récentes extensions portent sur leur implémentation efficace sur GPU. Une extension particulièrement efficiente basée sur un schéma de réduction-expansion de données est proposée dans [55]. Elle s'appuie sur une structure de données dite *HistoPyramids* pour un stockage compact de données et un requêtage efficient en cache. En pratique, les *marching cubes* et ses variantes sont implémentées au sein de shaders basées sur DIRECT-X ou OPENGL (ICARE-3D [35], FAST [56] ou HPMC [36] par exemple), ou de bibliothèques tels que VTK [37], ou CUDA toolkit [38].

\* \* \*



(1) maillage résultant sans et avec post-traitement

Figure 2.7: Artéfacts de voxelisation sur le maillage [39].

REMARQUE. À l'instar des méthodes de décomposition par grille, les *marching cubes* induisent des artefacts de discréttisation, dans le sens où des traces de découpage en voxel restent visibles (voir figure 2.71). Cela engendre des mailles à faible ratio d'aspect voire dégénérées. Bien que ça reste utilisable en visualisation, le maillage obtenu n'est pas directement utilisable en calcul numérique, et nécessite une étape de remaillage. C'est ce que nous allons justement voir dans la section suivante.

## 2.2 ADAPTATION DE LA SURFACE

Nous avons vu comment extraire une surface triangulée à partir d'un domaine définie explicitement ou implicitement. Bien qu'elle soit une bonne approximation de la surface idéale, les mailles résultantes sont trop étirées, ou présentent des effets de grille. En fait, ils ne sont pas directement utilisables en calcul numérique sans post-traitement. Le problème est que les schémas numériques nécessitent une discrétisation assez régulière pour converger correctement. Ici, nous montrons justement comment la qualité du maillage impacte la performance des solveurs numériques. Ensuite nous donnons une synthèse des techniques pour l'adaptation de surfaces triangulées.

BUT. En fait, le maillage issu d'un noyau de triangulation (DELAUNAY ou MARCHING CUBES par exemple) peut être auto-intersectée ou contenir des trous<sup>5</sup>, et peut nécessiter une réparation (ce qui est hors de notre cadre). De plus, il est souvent mal échantillonné, avec des mailles plates ou étirées, voire dégénérées. Ici, le but est d'adapter la surface triangulée de manière à réduire l'erreur d'une solution numérique ou bien l'erreur de la surface elle même, tout en garantissant une bonne qualité des mailles.

### 2.2.1 Anisotropie et qualité d'un maillage

En calcul numérique, la notion de qualité d'un maillage est relative car elle dépend de l'équation à résoudre (diffusion, écoulements, choc etc.). Elle est souvent définie par une norme sur la qualité individuelle des mailles.

#### 2.2.1.1 Impact sur les solveurs numériques

En fait, la qualité du maillage influe sur la précision et la vitesse de convergence des solveurs numériques, outre le fait qu'il doit être une bonne approximation du domaine.

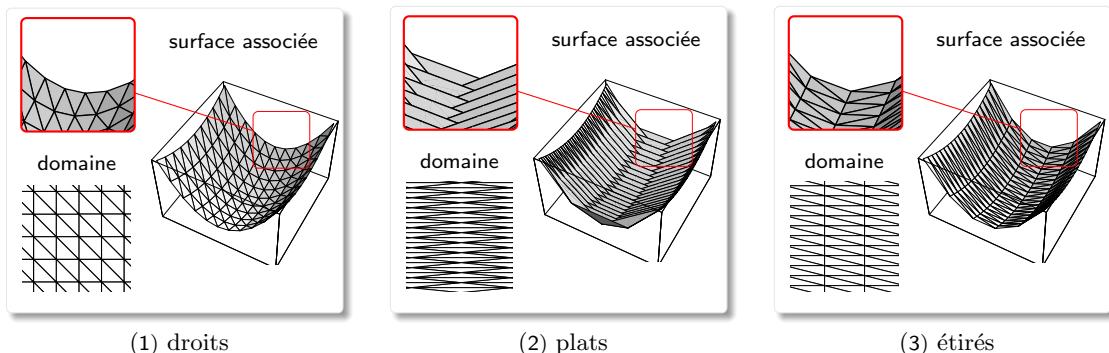


Figure 2.8: Influence de l'aspect des mailles sur l'interpolation d'une solution numérique en 2D [68].

- *précision* : l'erreur d'interpolation locale d'une solution numérique sur une maille  $K$  est souvent majorée par un facteur relatif au ratio d'aspect de  $K$  (et donc sa qualité) [69]. Elle dépend de la norme utilisée. En norme  $H^1$  par exemple<sup>6</sup>, on a :

$$\|\mathbf{u} - \Pi_h \mathbf{u}\|_{H^1(K)} \leq c \cdot q[K] \cdot |\mathbf{u}|_{H^1(K)}, \text{ avec } q[K] = \frac{h^2}{2r} \text{ et } \begin{cases} r : \text{rayon du cercle inscrit de } K \\ h : \text{le diamètre de } K \\ c : \text{facteur indépendant de } K \end{cases}$$

Ainsi l'erreur de  $\mathbf{u}$  dépend aussi de la forme de la maille, pas uniquement de la taille de ses arêtes. Pour se donner une idée intuitive, l'influence de la forme des mailles sur l'interpolation de  $\mathbf{u}$  dans

<sup>5</sup>Ainsi la surface triangulée n'est donc plus une variété

<sup>6</sup>En fait l'erreur d'une solution numérique est souvent estimée ou majorée par le biais d'une norme.

Ici, la norme  $H^1$  d'une fonction  $f : \Omega \rightarrow \mathbb{R}$  est définie par  $\|f\|_{H^1(\Omega)} = (\|f\|_{L^2(\Omega)}^2 + \sum_{i=0}^d \|\partial_{x_i} f\|_{L^2(\Omega)}^2)^{1/2}$ . Elle fait intervenir la norme  $L^2$  définie par :  $\|f\|_{L^2(\Omega)} = (\int_{\Omega} |f(x)|^2 dx)^{1/2}$ .

le cas 2D est donnée à la figure 2.8. Elle reprise de l'exemple dans [68]. Sur cet exemple, on voit que  $\mathbf{u}$  est mieux interpolée avec des mailles droites en (1), pas trop mal avec des mailles étirées en (2). Par contre, elle est pire avec des mailles plates en (3). Sinon, l'intégration numérique de quantités relatives à un point  $p$  dépend de la discréétisation du voisinage de  $p$  : sa précision dépend du degré de  $p$  et des angles des mailles incidentes à  $p$ . Par exemple, c'est le cas de la quadrature des cotangents [1] (§3.3.4) pour le calcul du laplacien sur une surface<sup>7</sup>. Ici, plus les mailles incidentes à  $p$  sont régulières, plus précise elle sera [62].

- *performance.* Si la solution numérique est calculée par une méthode des éléments finis, alors on a une équation  $\mathbf{AX} = \mathbf{B}$  à résoudre, où  $\mathbf{A}$  et  $\mathbf{B}$  sont des matrices carrées de taille  $n$ , et  $n$  est le nombre de points du maillage. Dans ce cas, la vitesse de convergence des solveurs itératifs (gradient conjugué, Jacobi) pour la résolution de cette équation dépend du conditionnement<sup>8</sup>  $\kappa_A$  de la matrice  $\mathbf{A}$ . Ainsi le temps de calcul est proportionnel à  $\kappa_A$ . Comme  $\mathbf{A}$  est symétrique, alors  $\kappa_A$  est juste le rapport des valeurs propres maximale et minimale  $\lambda_{\max}$  et  $\lambda_{\min}$  de  $\mathbf{A}$ . Il se trouve que  $\lambda_{\max}$  (et donc  $\kappa_A$ ) dépend de la forme des mailles [68]. En effet on a :

$$\kappa_A = O(\lambda_{\max}) = O(d \|q[\mathbf{K}]\|_\infty), \text{ avec } q[\mathbf{K}] = \frac{\sum_{i=1}^3 \ell_i^2}{4\sqrt{3}|\mathbf{K}|} \quad (2.2)$$

où  $d$  désigne le degré maximal des points du maillage, et  $\ell_i$  la longueur de la  $i^e$  arête de  $\mathbf{K}$  [68].

### 2.2.1.2 Forme et alignement optimal

Rappelons que l'optimalité d'un maillage  $\mathcal{T}_h$  dépend de l'équation à discréétiser. En fait, le maillage optimal celui qui suit mieux les variations de la solution numérique  $\mathbf{u}$  sur le domaine ou sur la surface. Ainsi c'est celui dont la densité de points et l'alignement des mailles sont adaptées au gradient de  $\mathbf{u}$ .

Table 2.1: Mesures usuelles de forme d'un triangle.

nomenclature	formule <sup>a</sup>	valeurs	idéale
min. angle [70]	$\theta_{\min}$	$[0, \pi]$	$\frac{\pi}{3}$
edge ratio [70]	$\frac{\ell_{\max}}{\ell_{\min}}$	$[1, \infty[$	1
radii ratio [70]	$\frac{R}{2r}$	$[1, \infty[$	1
aspect ratio [70]	$\frac{\ell_{\max}}{2\sqrt{3}r} = \frac{\ell_{\max}\sum_i \ell_i}{4\sqrt{3} \mathbf{K} }$	$[1, \infty[$	1
skewness [63]	$\max \left[ \frac{\theta_{\max} - \theta^*}{\pi - \theta^*}, \frac{\theta^* - \theta_{\min}}{\theta^*} \right]$	$[0, 1]$	0
orthogonality [63]	$\max \left[ 1 - \frac{\mathbf{n}_i \cdot \mathbf{v}_i}{\ \mathbf{n}_i \cdot \mathbf{v}_i\ } \right]$	$[0, 1]$	0
scaled jacobian [71]	$\frac{ J }{\max[\ell_i \ell_j]}$	$[-1, 1]$	$\frac{2\sqrt{3}}{3}$
frobenius ratio [69, 70]	$\frac{\sum_i \ell_i^2}{4\sqrt{3} \mathbf{K} }$	$[1, \infty[$	1

<sup>a</sup> Pour un triangle  $\mathbf{K}$ , on note :

- $\theta_{\min}$  et  $\theta_{\max}$  : angles minimal et maximal de  $\mathbf{K}$ .
- $\ell_{\min}$  et  $\ell_{\max}$  : longueurs minimale et maximale des arêtes de  $\mathbf{K}$ .
- $r$  et  $R$  : rayons des cercles inscrits et circonscrits à  $\mathbf{K}$ .
- $\mathbf{n}_i$  : vecteur normal à l'arête  $e_i$  de  $\mathbf{K}$ .
- $\mathbf{v}_i$  : vecteur reliant le barycentre de  $\mathbf{K}$  au milieu de l'arête  $e_i$ .
- $J$  : matrice jacobienne de  $\mathbf{K}$  avec  $J = (\vec{AB}, \vec{AC})$  en 2D.

<sup>7</sup>En fait, il s'agit de l'opérateur de l'opérateur de LAPLACE-BELTRAMI qui généralise le laplacien au cas des variétés. Il est impliqué dans les noyaux de paramétrisation, de segmentation, de reconstruction ou de lissage de variétés. Elle consiste en une quadrature de l'intégrale de la divergence du gradient au voisinage  $S$  du point  $p$  normalisée par l'aire de  $S$ , et implique les angles opposées à chaque arête incidente à  $p$ .

<sup>8</sup>Le conditionnement d'une matrice  $A$  est le nombre  $\kappa_A = \|A\|^{-1}\|A\|$ , où  $\|\cdot\|$  désigne la norme de FROBENIUS.

FORME. Une synthèse des mesures de forme d'un simplexe est donnée à la table 2.1. Elles sont basées sur les angles, les arêtes, l'aire ou encore la matrice jacobienne associée à une maille. Une analyse comparative de certaines d'entre elles est donnée dans [70], elles sont néanmoins plus ou moins équivalentes dans le sens où elles visent à discriminer les simplexes plats ou étirés des simplexes quasi-équilatéraux. Pour une mesure de forme normalisée  $q$  telle que 1 correspond au triangle idéal, l'adaptation consiste à maximiser la norme  $\|q\|_{p \in \mathbb{N}}$ . Sur cette table,

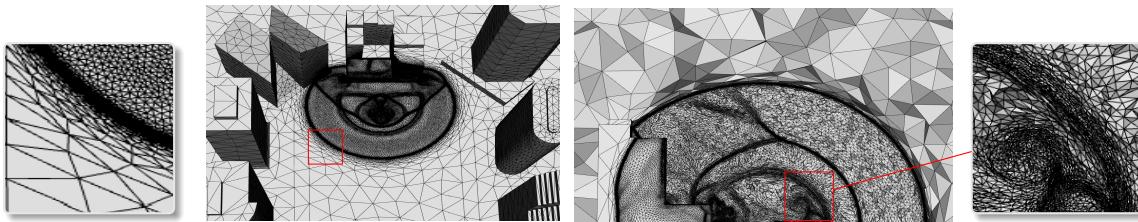
- **min-angle** est la mesure la plus simple et la plus intuitive. Ici, le remaillage vise à construire une triangulation de DELAUNAY du volume ou de la surface (§2.1.1). Elle convient si la solution numérique est suffisamment régulière (équations elliptiques par exemple), en vertu de la *condition d'angle* sur la convergence de fonctions quadratiques [72]. Néanmoins elle ne tient pas compte de la taille des simplexes définies par le diamètre<sup>9</sup>  $h$  du maillage  $\mathcal{T}_h$ .
- les **edge-radii-aspect ratio** quantifient l'aplatissement des mailles, tandis que le **skewness** et l'**orthogonality** mesurent leur déviation angulaire par rapport à un simplexe équilatéral.
- les **scaled-jacobian** et **frobenius ratio** sont basées sur la matrice jacobienne  $J$  de  $K$  qui encode les informations relatives au volume, la forme et l'orientation du simplexe. En effet une décomposition en valeurs singulières de  $J$  permet de retrouver les matrices associées à ces quantités [73] §3. Elles mesurent la déviation de  $K$  par rapport au simplexe idéal sur ces critères.

En pratique, des implémentations numériquement robustes de ces mesures existent au sein de bibliothèques comme CGAL [9] ou Verdict [64].

OPTIMALITÉ. In fine, toutes ces mesures sont purement géométriques et décorrelées de la solution numérique  $\mathbf{u}$ . Elles définissent la triangulation idéale comme celle constituée de mailles quasi-équilatérales. En fait, ce n'est vrai que si  $\mathbf{u}$  varie de manière isotrope, c'est-à-dire uniformément dans toutes les directions de  $\Omega$  (équation de la chaleur par exemple). Dans le cas anisotrope, une triangulation idéale est celle dont la répartition des points, la forme et l'étirement des mailles sont adaptées à l'erreur de la solution numérique, comme sur les deux exemples de la figure 2.9. En fait il s'agit de la triangulation dont la densité des points croît selon la courbure locale de  $\mathbf{u}$ , et dont les mailles sont alignées avec les directions de la hessienne<sup>10</sup> de  $\mathbf{u}$  sur chaque point<sup>11</sup>. Ainsi une bonne mesure de qualité doit encoder trois critères :

- une taille d'arêtes qui dépend des valeurs propres de la hessienne de  $\mathbf{u}$  en ses sommets.
- un alignement par rapport aux vecteurs propres de la hessienne de  $\mathbf{u}$  en ses sommets.
- un facteur de gradation borné<sup>12</sup>.

En fait, elle est souvent issue de l'une des mesures de la table 2.1, mais néanmoins modifiée pour tenir compte des informations fournies par un estimateur d'erreur basé sur les hessiennes de  $\mathbf{u}$  [74–77].



(1) densité d'onde de choc (surface)

(2) densité d'onde de choc (volume)

Figure 2.9: Exemples de triangulations optimales en mécanique des fluides [78].

<sup>9</sup>Le diamètre d'un triangle ou d'un maillage est la longueur de sa plus longue arête.

<sup>10</sup>La matrice hessienne  $H_u = \nabla^2 u$  de  $u$  sur un point  $p$  est une matrice symétrique définie positive décrivant la variation du gradient de  $u$ , et donc les coefficients sont formés par les dérivées secondes de  $u$ .

<sup>11</sup>Il a été montré dans [69] que l'erreur d'interpolation de  $u$  en norme  $H^1$  décroît en fonction de l'angle entre l'orientation d'une maille  $K$  et celle de  $u$ . De plus, si les mailles sont alignées avec  $u$  et que leur ratio d'aspect est borné par  $|\lambda_1/\lambda_2|^{1/2}$ , avec  $\lambda_i$  les valeurs propres de  $H_u$ , alors l'erreur de  $u$  ne dépend plus de l'angle maximal de  $\mathcal{T}_h$ .

<sup>12</sup>Il s'agit d'une borne sur la variation de tailles entre arêtes incidentes. En calcul numérique, il est important d'avoir une variation graduelle des tailles d'arêtes sur le domaine ou la surface, afin de permettre des interpolations plus stables.

### 2.2.2 Approches basées sur une paramétrisation

PRINCIPE. La modification directe d'une surface triangulée est délicate dans la mesure où elle fait intervenir diverses quantités relatives à la géométrie différentielle. Afin d'obtenir une reconstruction précise de la surface, tout en améliorant la répartition des points et la qualité des mailles, les noyaux de remaillage implique le calcul de quantités différentielles telles que les géodésiques, la courbure locale etc. A ce titre, il existe une classe de méthodes qui consiste à plonger localement la surface dans un domaine planaire, remailler la région locale, puis de projeter le maillage ainsi obtenu dans  $\mathbb{R}^3$  par le biais d'un homéomorphisme bijectif  $\sigma : \mathbb{R}^2 \rightarrow \mathbb{R}^3$  que l'on appelle *paramétrisation* (voir figure 2.10).

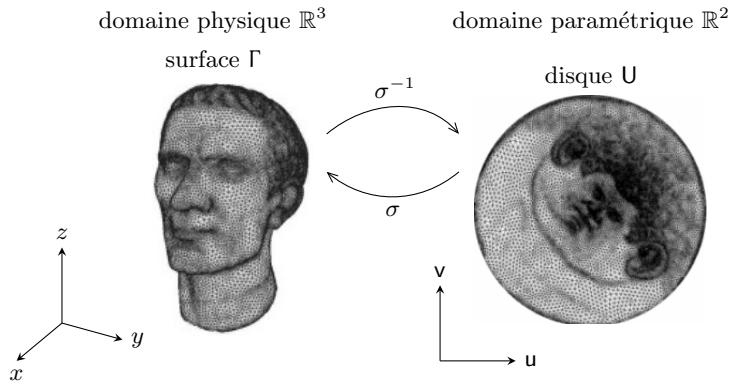


Figure 2.10: Mapping de la surface vers un plan par le biais d'une paramétrisation [CGAL]

L'intérêt de ces méthodes c'est qu'elles simplifient grandement les noyaux de remaillage puisqu'on travaille en 2D. Néanmoins elles présentent deux inconvénients :

- *distorsions* : la triangulation résultante dépend grandement de la qualité de la paramétrisation utilisée. En effet la projection d'une surface non triviale dans le plan paramétrique engendre inévitablement des distorsions d'aires, d'angles ou de longueurs. Ainsi des informations géométriques implicites peuvent être perdues en cours de route. Même si la paramétrisation minimise la distorsion dans un sens raisonnable, il est impossible de l'éliminer complètement.
- *coupe* : il n'est pas toujours possible de trouver une paramétrisation globale de la surface<sup>13</sup>. En fait, si la surface n'est pas homéomorphe à un disque, alors il faudrait une coupe géométrique tel que chaque patch puisse être paramétré séparément. Dans le cas d'une surface fermée, il faudrait maintenir une correspondance entre les points de part et d'autre des frontières des patchs. En effet il s'agit des mêmes points mais sont juste dupliqués.

Les avancées récentes en paramétrisation de surface permettent d'obtenir des algorithmes de remaillage robustes basées sur les transformations harmoniques, conformes, ou barycentriques [119–121]. Les techniques de paramétrisation impliquées dans ces algorithmes sont décrites dans [122].

### 2.2.3 Approches variationnelles

PRINCIPE. Elles regroupent les approches d'adaptation visant à un ré-échantillonnage **global** des points de la surface ou du domaine par un calcul variationnel, puis en triangulant le nuage de points ainsi obtenu. Dans ce cadre, l'adaptation de maillage est formulé comme un problème d'optimisation qui consiste à minimiser une énergie définie sur  $\Omega$ . Étant donné une densité qui définit le nombre de points par unité de surface, elle vise à calculer la répartition optimale des points conformément à cette densité. En fait c'est le choix de l'énergie à minimiser qui va distinguer les différentes approches [65, 66, 79–82]. L'approche la plus courante est celle basée sur les partitions de Voronoï [82–88] décrit à la définition 27. C'est ce que nous allons développer ici.

<sup>13</sup>cela dépend du genre de la surface (ou communément "genus" en anglais)

**Définition 27** (PARTITION DE VORONOÏ). Soit  $\Omega$  un ouvert de  $\mathbb{R}^3$ , et  $\{z_k\}_1^n$  un ensemble de points de  $\Omega$  appelés germes. Une partition de Voronoï est une partition de  $\Omega$  en  $n$  ouverts  $\{C_i\}_{i=1}^n$  tel que tout point de chaque  $C_i$  est plus proche de son germe  $z_i$  que d'un autre germe  $z_j$ , à savoir :

$$C_i = \{p \in \Omega \mid d(p, z_i) < d(p_k, z_j), \text{ pour tout } j \neq i\} \quad (2.3)$$

où  $d$  est une distance qui dépend d'une densité de points  $\rho : \Omega \rightarrow \mathbb{R}$ .

La partition est dite centroidale si les germes  $z_i$  coïncident avec leur centre de masse  $c_i$  de  $C_i$  :

$$z_i = c_i = \frac{\int_{C_i} x \rho[x] dx}{\int_{C_i} \rho[x] dx} \quad (2.4)$$

Notons indistinctement  $\Omega \subset \mathbb{R}^3$  le domaine ou la surface à trianguler. Ici, l'idée est de rééchantillonner  $\Omega$  en créant une partition  $\{C_i\}_{i=1}^n$  de  $\Omega$  telle que les points coïncident avec les germes des  $C_i$  comme illustré à la figure 2.11. Elle se formule par le problème d'optimisation suivant :

$$\min_{X_i} E[X_i] = \min_{X_i} \sum_{i=1}^n \int_{C_i} d(x, z_i) dx, \text{ avec } \begin{cases} X_i = \{C_i\}_1^n, \{z_i\}_1^n, \\ C_i \text{ est une partition de } \Omega, \\ z_i \text{ sont les germes des } C_i. \end{cases} \quad (2.5)$$

où  $d$  est une distance qui dépend de la densité  $\rho$  et précisément du fait qu'elle soit isotrope ou non.

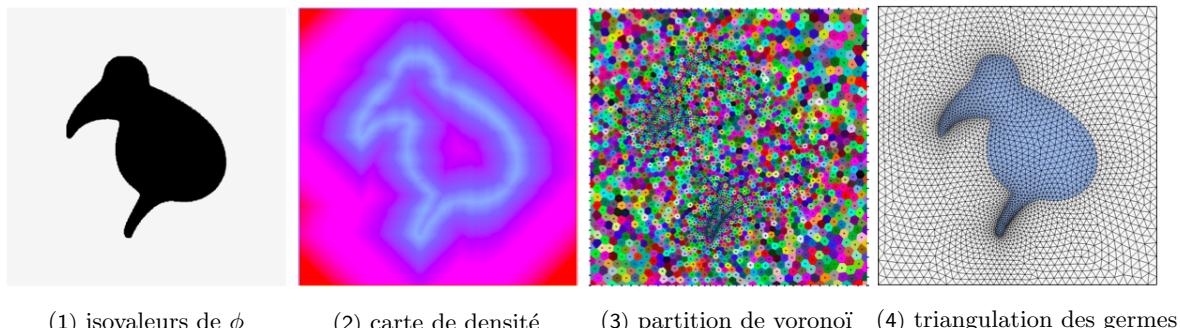


Figure 2.11: Triangulation de domaine implicite par une approche variationnelle [89]

### 2.2.3.1 Répartition des points

Dans le cadre de l'adaptation de maillage, la densité  $\rho$  va jouer un rôle important puisqu'elle va dicter la manière de répartir les points sur tout  $\Omega$ . Elle peut être déduite de la solution numérique calculée sur  $\Omega$ , ou d'une métrique dépendante de la surface telle que la courbure ou la largeur locale (LFS). En effet la triangulation extraite de la partition de Voronoï va dépendre du fait que  $\rho$  varie de manière isotrope ou anisotrope. Dans ce cadre,  $\rho$  redéfinit le voisinage continu de chaque point selon la taille d'arête souhaitée dans chaque direction de  $\Omega$  comme sur la figure 2.12. En fait, elle définit une variété riemannienne au sens de la définition 13.

- Dans le cas isotrope, la densité d'un point  $p_i$  se réduit à un scalaire  $\lambda_i$  et la distance utilisée dans (2.5) est donnée par :  $d(p_i, p_j) = \lambda_i \langle \mathbf{v}, \mathbf{v} \rangle^{1/2}$ , où  $\mathbf{v} = p_j - p_i$ .
- Dans le cas anisotrope, la densité s'exprime en fonction d'un tenseur métrique associé à chaque point de  $\Omega$ . Il s'agit d'une matrice symétrique définie positive associé à point de  $\Omega$  et qui redéfinit localement le produit scalaire au voisinage de  $p$ . Ainsi, la distance utilisée dans (2.5) est donnée par :  $d(p_i, p_j) = \langle \mathbf{v}, g_{p_i} \mathbf{v} \rangle^{1/2}$  où  $\mathbf{v} = p_j - p_i$ .

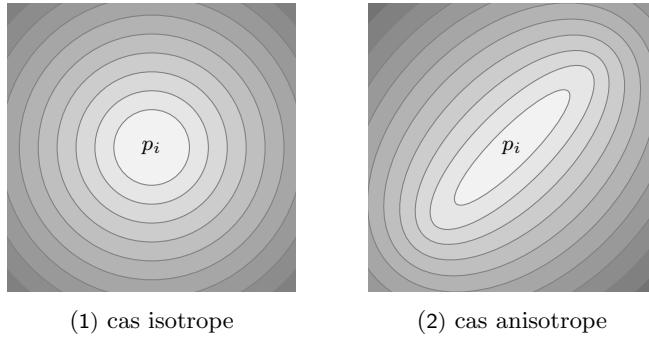


Figure 2.12: Voisinage d'un point en fonction de sa densité.

CONCRÉTEMENT. Les approches se différencient sur les algorithmes impliqués dans la minimisation de  $E$ . Concrètement, elle se fait usuellement par un algorithme de *clustering* [85], ou un algorithme de descente comme la relaxation de LLOYD [90] ou encore L-BGFS [91]. Concrètement, le domaine est décrit par une triangulation initiale, et il faudrait disposer d'une partition initiale  $\{C_i\}$  pour la minimisation de  $E$ . Par ailleurs, il faudrait pouvoir évaluer numériquement les intégrales dans l'expression de  $E$  et les centres de masses  $c_i$  de manière discrète. Leur approximation dépend de la méthode utilisée que l'on expliquera dans les deux paragraphes suivants.

### 2.2.3.2 Noyaux globaux

CLUSTERING. Souvent utilisé en simplification de maillages, il consiste à créer une partition de Voronoï dans lesquels les cellules  $\{C_i\}_{i=1}^n$  correspondent à des clusters de points de la triangulation initiale. Dans ce cas, les germes  $z_i$  sont choisis (aléatoirement ou selon un critère géométrique) parmi l'ensemble des points du maillage, et chaque cellule  $C_i$  associée à chaque  $z_i$  correspond à l'ensemble des volumes de contrôle  $I_j$  de chaque point de  $C_i$ . Ainsi la triangulation finale aura moins de points. Dans le cas isotrope, l'énergie à minimiser dans 2.5 est approchée par :

$$E[X_{i,j}] = \sum_{i=1}^n \sum_{j=1}^m \omega_j \langle \mathbf{v}, \mathbf{v} \rangle^2 + \sum_{j=1}^m a_j, \quad \left\{ \begin{array}{l} X_{i,j} = \{I_j\}, \{z_i\}, \mathbf{v} = z_i - \gamma_j \\ w_j = \int_{I_j} \rho(x) dx : \text{aire de } I_j \\ \gamma_j = w_j^{-1} \int_{I_j} x \rho(x) dx : \text{centre de masse de } I_j \\ a_j = \int_{I_j} \rho(x) \|x - \gamma_j\| dx : \text{distance moyenne} \\ \text{de } \gamma_j \text{ avec les points de } I_j \end{array} \right. \quad (2.6)$$

où les termes de (2.6) peuvent être calculés sur chaque point  $p_j$  par la formule de quadrature des cotangents [1] (§3.3.4). Dans le cas anisotrope, l'énergie s'écrit :

$$E[X_{i,j}] = \sum_{i=1}^n \sum_{j=1}^m \langle \mathbf{v}, g_{p_j} \mathbf{v} \rangle. \quad (2.7)$$

L'algorithme de clustering proprement dit se base sur des tests des points situés sur les frontières des clusters. En effet chaque arête  $e \in C_i \cap C_j$  est incident à deux items  $I_a \in C_i$  et  $I_b \in C_j$ . Dans ce cas, on évalue  $E$  dans la configuration initiale de  $I_a$  et  $I_b$ , puis dans les deux cas de figures où on place  $I_a$  dans  $C_j$ , et  $I_b$  dans  $C_i$ . On garde la configuration qui minimise localement  $E$ . En bouclant sur une liste d'arêtes frontières, cela permet de minimiser  $E$  de manière incrémentale. Une décomposition plus fine et un stockage mémoire cumulatif des termes de  $E$  permet d'accélérer le calcul mais le principe est le même. Bien qu'elle soit conceptuellement simple, cette approche est garantie de converger [85].

OPTIMISATION. Cette fois la partition optimale est vue comme un point critique de  $E$ . Ainsi elle est résolue par des méthodes d'optimisation non-linéaire de type Newton. Dans ce cadre, elle correspond à la partition  $X^* = \{C_i\}^*, \{z_i\}^*$  pour lequel  $\|\nabla E(X^*)\| = 0$ <sup>14</sup>. L'approche consiste à résoudre le système

<sup>14</sup>Ici, le gradient de  $E$  s'exprime par  $\partial_{x_i} E = 2m_i(x_i - z_i)$ , avec  $m_i = \int_{C_i} \rho(x) dx$  la masse de  $C_i$

$H\mathbf{d}\mathbf{x} = -\nabla E$  afin de déterminer le pas de déplacement  $\mathbf{d}\mathbf{x}$  à chaque itération,  $H$  étant la hessienne de  $E$ . L'inconvénient de cette méthode c'est que le calcul et la mise à jour de la hessienne est très coûteuse. Ainsi en pratique, une méthode quasi-Newton telle que L-BGFS<sup>15</sup> est souvent préférée [91]. L'avantage est double : seules la valeur de la fonctionnelle et de son gradient sont évaluées et n'implique que des multiplications matrice-vecteurs d'une part, et seuls quelques vecteurs de  $H$  sont stockées en mémoire d'autre part [92].

## 2.2.4 Approches locales

Elles regroupent les méthodes qui consistent à adapter la surface triangulée par des noyaux locaux jusqu'à convergence en erreur ou en qualité. Dans ce cas, les noyaux consistent au ré-échantillonnage de la surface et à la régularisation des mailles. Notons que d'un point de vue combinatoire, adapter une surface est plus simple qu'adapter un volume puisque les opérations topologiques<sup>16</sup> portent sur des triangles et non sur des tétraèdres. Néanmoins les noyaux surfaciques impliquent l'estimation de quantités relatives à la géométrie différentielle<sup>17</sup> et cela sur une surface discrète. Ainsi, leur précision influe sur la rapidité des algorithmes. Notons qu'il existe une vaste littérature concernant les méthodes incrémentales, chacune se différenciant sur le but de l'adaptation et l'application cible.

### 2.2.4.1 Noyaux locaux

**PRINCIPE.** En optimisation, un algorithme de recherche locale consiste à chercher la solution optimale par exploration de l'espace de solutions. Partant d'une solution candidate, il consiste à se déplacer itérativement vers une solution voisine jusqu'à tomber sur un optimum, ou si un critère d'arrêt est satisfait (timeout ou pas d'amélioration depuis un certain nombre d'itérés par exemple). Le remaillage local suit ce schéma. Partant d'un maillage qui est une assez bonne approximation de la surface, il consiste à appliquer une séquence d'opérations locales afin d'améliorer la répartition des points et la qualité des mailles. Le traitement est effectué jusqu'à ce qu'un seuil sur l'erreur d'une solution numérique ou un nombre maximum d'itérés a été atteint. Ici le point subtil consiste à trouver une séquence judicieuse d'application des noyaux pour accélérer la convergence. Néanmoins aucune combinaison particulière n'a été prouvée comme optimale dans la littérature.

- *raffinement* : il vise à enrichir la triangulation en insérant successivement des points de STEINER dans les régions sous-échantillonées de la surface  $\Gamma$ , conformément à une densité prescrite. Un noyau de Delaunay peut être utilisé à cette fin, néanmoins l'extension au cas des variétés n'est pas triviale contrairement au cas planaire [123, 124]. Ainsi des noyaux plus simples basés sur une dissection d'arêtes sont souvent utilisés en pratique [125].
- *simplification* : elle vise à supprimer les points dans les régions trop denses de la surface. Pour cela, on recourt usuellement à un noyau de suppression d'arêtes ou ses variantes [1] (§7.2.1). À noter que ce noyau requiert des précautions préalables, sans quoi la triangulation résultante peut être invalide<sup>18</sup>.
- *bascule d'arêtes* : elle vise à régulariser le degré des points de la triangulation, ou bien à améliorer la qualité des mailles en changeant leur connectivité<sup>19</sup>
- *lissage* : il vise à optimiser la qualité des mailles en relocalisant les points. Il consiste à déplacer chaque point  $p$  vers une position optimale  $p^*$  sans modifier la connectivité des mailles. En fait il

<sup>15</sup>L-BGFS pour *Limited-memory Broyden-Fletcher-Goldfarb-Shanno*

<sup>16</sup>Par exemple le découpage de mailles, suppression de points, bascules d'arêtes, reconnexion de mailles.

<sup>17</sup>Comme le calcul de géodésiques, aires de triangles courbes, courbures locales, tenseurs métriques, voire des connexions pour le transport parallèle de vecteurs.

<sup>18</sup>En fait, il n'est applicable que si les *conditions de lien* sont respectées [32].

- si  $\mathbf{p}$  et  $\mathbf{q}$  sont des points-frontière, alors  $\mathbf{e}$  est une arête-frontière.
- l'intersection des boules topologiques de  $\mathbf{p}$  et  $\mathbf{q}$  ne contient que les deux points opposés à  $\mathbf{e}$ .
- les vecteurs normaux des mailles incidentes à  $\mathbf{q}$  gardent la même orientation avant et après suppression de  $\mathbf{p}$ .

<sup>19</sup>À l'itéré  $t$ , on bascule l'arête commune entre deux mailles :

- si  $\partial_t \|\chi\|_2 \leq 0$  avec  $\chi$  l'écart du degré d'un point par rapport au degré optimal.
- ou si  $\partial_t \|\frac{1}{q}\|_1 \cdot \partial_t \|\frac{1}{q}\|_\infty \leq 0$  où  $q$  mesure la qualité de mailles.

y a plusieurs choix pour  $p^*$ , mais la stratégie la plus utilisée est certainement le *lissage laplacien* ou ses variantes [33]. Dans ce cas, le point optimal est la moyenne pondérée des voisins<sup>20</sup>, à savoir  $p^* = \sum_{i=1}^n \omega_i p_i$ ,  $n$  étant le nombre de voisins de  $p$ , et  $\omega_i \in [0, 1]$  des poids<sup>21</sup> quantifiant la contribution de chaque voisin au déplacement de  $p$ . Le noyau est ensuite appliqué itérativement jusqu'à ce qu'un seuil de qualité soit atteint. Bien que simple et souvent efficace, il ne garantit pas une amélioration stricte de la qualité des mailles. Ainsi des noyaux plus compliqués existent dans la littérature. Ils sont basés sur la minimisation d'une fonction liée à la qualité des mailles incidentes à  $p$  pour trouver  $p^*$ , par le biais de techniques d'optimisation non-linéaires (voir [126–128] par exemple).

#### 2.2.4.2 Reconstruction de la surface

BUT. En fait, les noyaux ci-dessus nécessitent de pouvoir reconstruire localement la surface sur une maille ou au voisinage d'un point. Pour le raffinement, le point est projeté sur une courbe sous-tendue par l'arête à couper. Pour la simplification, le point résultant  $q$  est repositionné de manière à minimiser l'erreur induit par la suppression du point  $p$ . Pour le lissage, le point optimal est souvent calculé dans le plan tangent du point à déplacer, en absence d'une paramétrisation. Ainsi il doit être projeté sur une surface reconstruite au voisinage de  $p$ . En fait, il existe plusieurs manières de reconstruire les courbes et surfaces dont les plus usuelles sont :

- *interpolation* : ils permettent de retrouver la surface  $\Gamma$  sur une maille  $K$  ou une courbe  $\gamma$  sur une arête, en interpolant des points de contrôle répartis sur  $\gamma$  ou  $\Gamma$ . En pratique, on utilise souvent des B-splines à poids uniforme : les courbes et triangles de BÉZIERS. Un triangle de BÉZIERS est une surface obtenue par interpolation de points de contrôle et délimitée par trois courbes splines paramétrées. Ainsi tout point  $p$  de la surface passant sur  $K$  est interpolé comme suit :

$$\Gamma_K^n(u, v) = \sum_{\substack{0 \leq i, j, k \leq n \\ i+j+k=n}} \mathbf{B}_{ijk}^n b_{ijk}, \text{ avec } \begin{cases} (b_{ijk})_{i=1}^n : \text{points de contrôle du patch,} \\ \mathbf{B}_{ijk}^n : \text{polynômes de Bernstein de degré } n, \\ (u, v) : \text{ coordonnées barycentriques de } p \text{ dans } K. \end{cases} \quad (2.8)$$

Notons que le choix des points de contrôle est libre : c'est ce qui va différencier les approches. En pratique, il dépend des contraintes que l'on veut respecter sur la triangulation (unicité des plans tangents aux sommets des mailles par exemple) [129, 130]. Sans contrainte, ils peuvent être naïvement calculés par le biais de l'algorithme de Casteljau.

- *approximation* : une manière de reconstruire la surface consiste à l'approcher au voisinage d'un point  $p_i$  par un surface quadrique d'équation implicite  $\Gamma_i(x, y, z) = ax^2 + bxy + cy^2 - z = 0$ . Ainsi déterminer  $\Gamma_i$  revient à minimiser  $\sum_{k=1}^n (ax_k^2 + bx_ky_k + cy_k^2 - z_k)^2$ ,  $n = |N_i|$ , ce se fait usuellement par une méthode des moindres carrés. Concrètement, cela consiste à résoudre l'équation matricielle suivante :

$$AX = B \Rightarrow \begin{pmatrix} u_1^2 & u_1v_1 & v_1^2 \\ \vdots & \vdots & \vdots \\ u_n^2 & u_nv_n & v_n^2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} d_1 \\ \vdots \\ d_n \end{pmatrix} \quad (2.9)$$

où  $(u_k, v_k)$  correspond aux coordonnées des voisins  $p_k$  de  $p_i$  dans le repère local au plan tangent de  $p_i$ , tandis que  $d_k$  est l'écart entre  $p_k$  et son projeté sur ce plan tangent. Si  $n > 3$  alors le système est surdéterminé. Dans ce cas, l'équation 2.9 est reformulé en  $A^\top AX = A^\top B$ , puis résolu en utilisant une méthode d'inversion matricielle de Gauss [107] (§3.1).

<sup>20</sup>En pratique, la position optimale est relaxée par un facteur  $\lambda$  tel que  $p^* = (1 - \lambda)p + \lambda p^*$  avec  $\omega \in [0, 1]$ .

<sup>21</sup>Il peut s'agir d'un ratio de longueurs d'arêtes, d'angles ou d'aires des mailles incidentes à  $p$ .

### 2.2.4.3 Cartes de tailles

PRINCIPE. Afin de contrôler la densité des points ainsi que l'étirement des mailles, il faudrait disposer d'une structure qui permette de prescrire les tailles d'arêtes au voisinage d'un point pour une direction donnée  $\mathbf{v}_i$ . Pour cela, une manière usuelle consiste à recourir aux tenseurs métriques. Un tenseur métrique associé à un point  $p$  est une matrice symétrique définie positive  $g_p$  qui définit un produit scalaire local pour le calcul de distances et d'angles dans le plan tangent de  $p$ . Il peut être représenté par la boule formée par les points qui sont à distance unité dans cet espace métrique comme sur la figure 2.13. Dans la base canonique de  $\mathbb{R}^3$ , il s'écrit <sup>22</sup> :

$$g_p = P^{-1} \Lambda P, \text{ avec } \begin{cases} \Lambda = \text{diag}(h_i^{-2})_{i=1}^3 : \text{les valeurs propres de } g_p \\ P = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3) : \text{les directions principales de } g_p \\ h_i : \text{la taille d'arête prescrite en direction de } \mathbf{v}_i \end{cases} \quad (2.10)$$

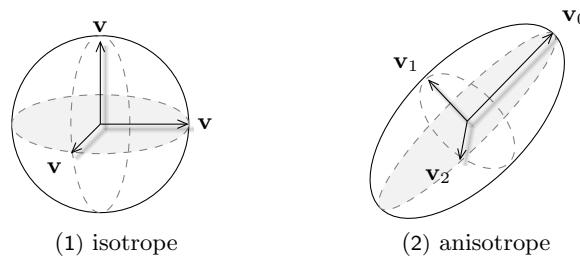


Figure 2.13: Boule d'un tenseur métrique.

CHAMP. En pratique, un champ de métriques est construit sur les points du maillage. Dans ce cadre, les  $h_i$  peuvent être relatives aux **courbures locales** ou aux valeurs propres de la hessienne d'une solution numérique  $\mathbf{u}$ . A chaque création ou déplacement d'un point, le tenseur associé est interpolé à partir de ses voisins. En pratique, cela est fait par le biais d'une *réduction simultanée* [93] ou d'une interpolation *log-euclidienne* [94]. La construction du champ fait intervenir un estimateur d'erreur, et il existe une vaste littérature à ce sujet [74–77]. En raison de sa simplicité, un estimateur en norme  $L^\infty$  est souvent utilisé dans les remailleur génériques [108, 95].

Dans ce cas précis, l'erreur relative à une maille  $K$  s'écrit :

$$\|\varepsilon(\mathbf{u})\|_{L^\infty(K)} \leq \frac{2}{9} \max_{x \in K} \max_{\mathbf{v} \in K} \langle \mathbf{v}, H_{\mathbf{u}}(x) \mathbf{v} \rangle \quad (2.11)$$

Étant donné un seuil d'erreur  $\varepsilon$ , le champ peut être construit via (2.10) en remplaçant les  $h_i$  par :

$$h_i = \max \left[ \min \left[ \left( \frac{9\varepsilon}{2|\lambda_i|} \right)^{1/2}, h_{\max} \right], h_{\min} \right]. \quad (2.12)$$

EN BREF. Le recours aux tenseurs métriques pour le contrôle des noyaux adaptatifs est devenu le standard en remaillage local pour la mécanique des fluides. En effet il permet d'obtenir des triangulations uniformes, adaptées isotropes ou anisotropes en redéfinissant le produit scalaire local à chaque point, tout en gardant les mêmes noyaux. Ainsi, bon nombre de remailleur sont basés sur ce concept tels que **yams** [108], **madlib** [96], **MMG** [95] ou **pragmatic** [199] pour en citer quelques-uns. Néanmoins d'autres alternatives existent dans le cadre général. L'approche décrite dans [131] consiste à plonger le domaine dans  $\mathbb{R}^6$  puis d'appliquer les noyaux sur cet hypersurface. Il ne permet néanmoins que l'adaptation de la triangulation aux courbures de la surface.

\* \* \*

<sup>22</sup>Dans ce cadre, les grandeurs géométriques deviennent :

- la longueur d'une courbe  $\gamma : [0, 1] \rightarrow \Gamma$  par  $\ell_h(\gamma) = \int_0^1 g_p(\dot{\gamma}[t], \dot{\gamma}[t])^{1/2} dt$ .
- l'aire d'une maille  $K$  par :  $|K| = \int_K (\det[g_x]^{1/2} dx)$ .
- la distance entre deux points  $p, q$  par :  $\ell_h(pq) = \inf_{\gamma} \ell_h(\gamma)$ , avec  $\gamma \in C^1([0, 1])$ .

### 2.3 SYNTHÈSE

En bref, nous avons vu que la génération de la triangulation initiale dépend de la représentation de la surface en entrée qui peut être explicite (nuage de points) ou implicite (fonction distance); mais qu'elle nécessite un traitement ultérieur (ou remaillage) dans les deux cas. À ce titre, nous avons également vu les différentes stratégies de remaillage d'une surface triangulée. Étant donnée une densité, les méthodes variationnelles fournissent la meilleure discréétisation possible en termes de répartition de points et de qualité des mailles, grâce à un rééchantillonnage complet de la surface. Souvent basées sur une routine de descente, elles sont néanmoins lentes à converger. Ainsi elles ne conviennent pas à une boucle numérique adaptative en raison du surcoût prohibitif : cela est d'autant plus vrai lorsque la solution numérique ne varie que graduellement à chaque pas de temps. Une approche de remaillage local permet de palier ce problème. En se basant sur des algorithmes de recherche locale, elle vise à adapter graduellement la surface triangulée par le biais de noyaux locaux, afin de satisfaire la densité prescrite. Néanmoins leur convergence n'est pas toujours garantie car on peut aisément tomber sur un minimum local<sup>23</sup> : à vrai dire c'est le choix des noyaux ainsi que leur contrôle qui va grandement influencer l'efficacité du remailleur<sup>24</sup>.

\* \* \*

---

<sup>23</sup>Cela implique une discréétisation moins régulière ou un nombre sous-optimal de points de Steiner comparé à une méthode variationnelle.

<sup>24</sup>en termes d'erreur d'interpolation ou de qualité de mailles

## **Part II**

---

# **Contributions**

## Design de noyaux surfaciques locality-aware.

Rappelons que notre but est d'accélérer la boucle numérique adaptative décrite à la page 11, en proposant des noyaux d'adaptation de maillages qui exposent une localité maximale requise par le hardware, malgré leur irrégularité intrinsèque. En fait cette contrainte est loin d'être triviale car elle limite significativement les choix algorithmiques dans la construction de noyaux, et donc leur efficacité. Dans ce chapitre, nous montrons comment nous conciliions ces contraintes de localité, tout en restant aussi efficaces que les noyaux de référence en termes de convergence en erreur ou en qualité de mailles.

**Contributions :**

- une projection de points basée sur l'application exponentielle,
- un noyau de lissage mixte diffusion-optimisation non linéaire,
- une preuve d'équivalence des mesures usuelles de gradation,
- un transport optimal de tenseurs métriques sur une variété.

Publications : un article [RL18] et une présentation [LR18].

3.1	Introduction . . . . .	54
3.1.1	Cadre et contraintes . . . . .	54
3.1.2	Démarche et contributions . . . . .	55
3.2	Design des noyaux . . . . .	56
3.2.1	Hypothèses et optimalité . . . . .	56
3.2.2	Pré-traitement . . . . .	57
3.2.2.1	Extraction des ridges . . . . .	58
3.2.2.2	Représentation de la topologie . . . . .	58
3.2.2.3	Orientation de la surface . . . . .	60
3.2.3	Reconstruction et projection . . . . .	61
3.2.3.1	Patch spline par maille . . . . .	62
3.2.3.2	Blending pour la continuité $G^1$ . . . . .	63
3.2.3.3	Notre opérateur de projection . . . . .	64
3.2.4	Recherche locale . . . . .	66
3.2.4.1	Raffinement . . . . .	66
3.2.4.2	Simplification . . . . .	67
3.2.4.3	Relaxation . . . . .	69
3.2.4.4	Lissage . . . . .	70
3.2.5	Stratégie complète . . . . .	78
3.3	Extension au cas anisotrope . . . . .	80
3.3.1	Répartition anisotrope des points . . . . .	80
3.3.1.1	Intérêt et principe . . . . .	80
3.3.1.2	Normalisation . . . . .	82
3.3.1.3	Gradation . . . . .	84
3.3.2	Transport de métriques . . . . .	87
3.4	Évaluation numérique . . . . .	91
3.4.1	Cadre, but et mesures . . . . .	91
3.4.2	Précision des noyaux . . . . .	92
3.4.3	Convergence . . . . .	96
3.5	Conclusion . . . . .	98

## 3.1 INTRODUCTION

### 3.1.1 Cadre et contraintes

**BUT.** Dans le cadre de simulations adaptatives impliquant un solveur numérique et un remailleur 3D, notre but est de fournir des noyaux surfaciques qui exposent une localité maximale requise par les architectures manycore, tout en restant aussi efficace que les noyaux de référence en termes de convergence en erreur ou en qualité.

Partant d'une surface triangulée uniforme et d'un budget de points, nous visons à fournir un maillage qui minimise l'erreur d'interpolation de la solution numérique calculée, ou l'erreur d'approximation de la surface elle même<sup>1</sup>, tout en maintenant un bon ratio d'aspect des mailles à la fois en contexte isotrope et anisotrope, comme illustré sur la figure 3.1.

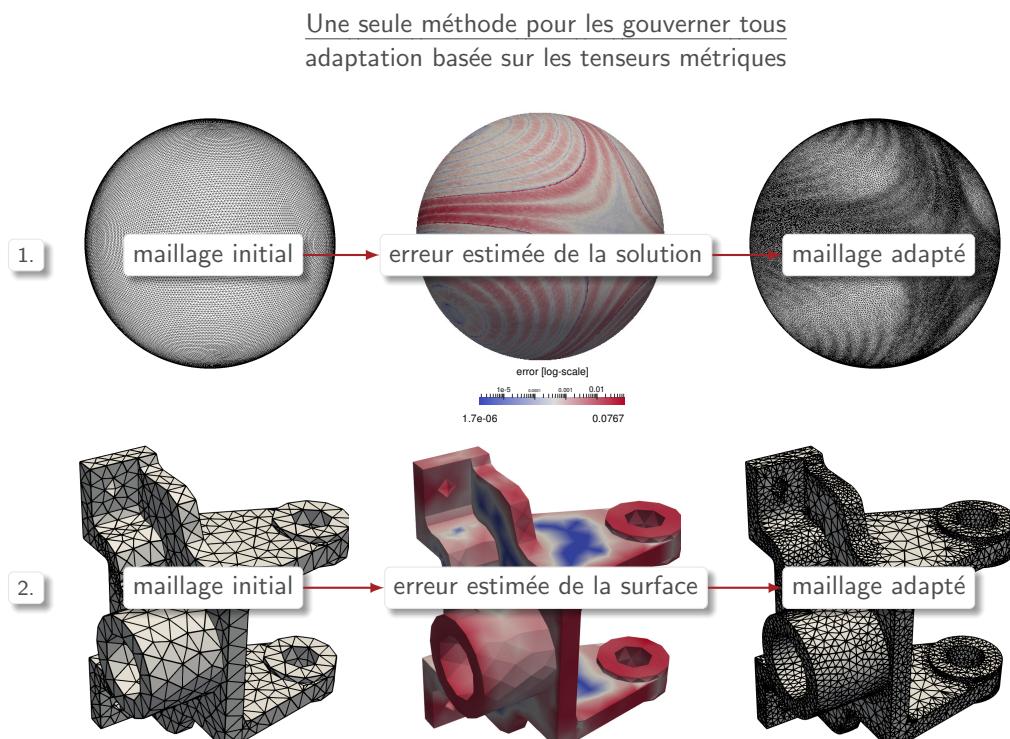


Figure 3.1: Les deux types d'adaptation gérées.

**TRAVAUX CONNEXES.** L'adaptation locale de surface ou volume est une thématique bien étudiée [11], et plusieurs outils robustes et open-source existent comme `tetgen`, `meshlab`, CGAL, MADLIB, MMGS-3D, ou GMSH [9, 96, 132–134]. Ainsi une question légitime concerne l'utilité de réinventer la roue. En fait, les noyaux impliqués dans ces remailleurs n'exposent pas suffisamment de localité requis par le hardware. En effet, chaque opération sur un point ou une maille ne devrait impliquer qu'un voisinage restreint, statique et borné. Notons que cette contrainte de localité est loin d'être triviale, et les noyaux les plus efficaces de l'état de l'art ne la respecte pas. À titre d'exemple :

- le recours aux **cavités dynamiques**<sup>2</sup> permet d'accélérer la convergence en qualité [123, 97]. En fait cela permet de supprimer les mailles distordues au voisinage d'un point donné, lors de l'insertion ou suppression de ce point. Néanmoins ces mailles ne peuvent pas être inférées statiquement.

<sup>1</sup>En fait dans notre cas, la répartition des points ainsi que l'alignement est guidé par un champ de tenseurs métriques défini sur les points du maillage. Si on veut adapter le maillage à l'erreur de la solution numérique  $\mathbf{u}$ , alors ce champ peut être extrait des hessiennes locales de  $\mathbf{u}$ . Si c'est à l'erreur de la surface elle-même, alors il est extrait des tenseurs de courbures. Enfin, les deux erreurs peuvent être gérées simultanément en utilisant l'intersection de tenseurs métriques.

<sup>2</sup>Comme les noyaux de Delaunay surfacique ou anisotrope, et le hybrid cavity kernel de LOSEILLE et al.

- le remaillage à base d'*atlas* fournit des surfaces bien échantillonées, avec une qualité comparable à celle issue d'une méthode variationnelle. Ici la surface est localement plongée dans un plan par le biais d'une paramétrisation, puis remaillée via des noyaux 2D. Néanmoins construire l'*atlas* à la volée implique de parcourir et figer un voisinage non prédefini de mailles à chaque fois [120].
- les calculs numériques sont plus stables sur des maillages quasi-structurés<sup>3</sup>. Relaxer les degrés des points est néanmoins fastidieuse due à de nombreux minima locaux. En fait les noyaux de référence (5-6-7 scheme, puzzle solving) s'appuient sur une séquence dynamique de basculement-refinement-suppression d'arêtes : les mailles impactées ne peuvent être inférées [120, 135, 136].

En fait, nous tentons de concilier deux contraintes antagonistes. Pour satisfaire la localité, on est constraint d'utiliser que de noyaux très basiques (pas de cavité dynamique, pas de plongement via un atlas, pas de séquence dynamiques d'opérations). Pour rester aussi efficient que l'état de l'art, on est constraint de recourir à des noyaux dynamiques afin de converger rapidement en termes d'approximation de la surface, de qualité de mailles ou bien d'interpolation de la solution numérique.

### 3.1.2 Démarche et contributions

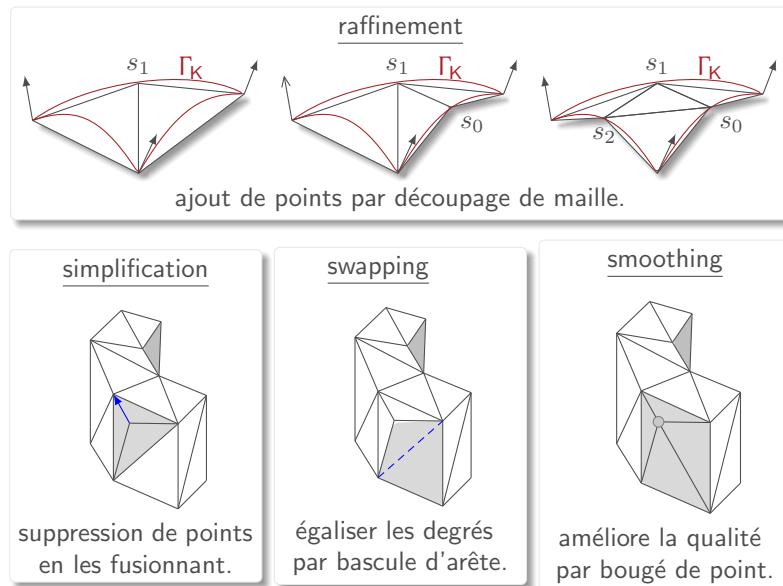


Figure 3.2: Nos noyaux statiques. Ici, le raffinement et la simplification vise à rééchantillonner la surface, tandis que la bascule et le lissage vise à la régulariser. En l'occurrence, ils n'impliquent qu'une maille, une paire de mailles ou le voisinage direct d'un point.

CONTRIBUTIONS. Dans ce cadre, nous visons à concilier les deux contraintes. Pour exposer une localité maximale, nous ne recourons que les **noyaux basiques** décrits à la figure 3.2, et nous évitons les features dynamiques décrites précédemment. Pour rester aussi efficient en termes de convergence, les noyaux que nous proposons s'appuient sur des primitives géométriques avancées. Ils incluent :

- Projection de points (raffinement, simplification et lissage).

Dans notre cas, la surface idéale n'est connue que sur les points du maillage, et nous ne disposons pas d'un atlas pour une paramétrisation locale. Ainsi, nous devons trouver une manière précise de placer les points sur cette surface idéale lorsqu'on coupe un' arête, qu'on fusionne deux points ou encore quand on déplace un point. Si le point est déjà sur le maillage alors ce n'est pas un problème puisque dans ce cas, nous pouvons utiliser des courbes de BÉZIERS, un PN-triangle ou une surface quadrique par exemple. Néanmoins s'il n'est pas sur le maillage, alors la situation n'est pas claire. Dans ce cas, nous devons d'abord le projeter sur le maillage, puis ensuite sur

<sup>3</sup>Autrement dit, un maillage où tous les points ont quasiment le même nombre de voisins : 6

la surface idéale, ce qui implique une perte inévitable de précision. Ici, nous proposons une manière de projeter les points qui s'appuie sur le calcul de courbes géodésiques via un opérateur de géométrie différentielle. Il assure que la perte de précision reste minime, et contribue de ce fait à accélérer la convergence du maillage en terme d'approximation de la surface.

- **Relocalisation de points** (lissage).

Pour réduire le nombre d'itérés, les noyaux de raffinement, de simplification et de bascule sont appliqués sans tenir compte de la qualité des mailles. Ainsi l'amélioration de la qualité du maillage est entièrement confiée au lissage. Il consiste à déplacer les points afin d'améliorer la forme des mailles incidentes. Néanmoins, déplacer un point implique inévitablement une déformation de la surface. Ici, nous proposons un noyau de lissage qui vise à améliorer la qualité tout en réduisant la dégradation de la surface. Il s'appuie sur un filtre de diffusion qui est d'abord appliqué au point courant, puis une routine d'optimisation qui est invoquée en cas d'échec. Il contribue ainsi à accélérer la convergence du maillage en qualité.

- **Transport de métriques** (raffinement et lissage).

En fait, les tailles d'arêtes souhaitées sont prescrites sur le maillage par le biais de tenseurs métriques. Ils permettent d'adapter le maillage aussi bien à l'erreur estimée de la solution numérique qu'à l'erreur estimée de la surface elle même, en contexte anisotrope comme montré sur la figure 3.1. Pour chaque point, son tenseur métrique donne la taille souhaitée de des arêtes dans toutes les directions qui lui sont incidentes. Ainsi, si un point  $p$  est créé ou déplacé, alors son tenseur métrique doit être interpolé à partir de ses voisins. Néanmoins, les interpolations répétées de métriques implique inévitablement une perte d'anisotropie. Pour le réduire, on peut déplacer tous les tenseurs voisins sur  $p$  avant de considérer leur moyenne (si nécessaire). Néanmoins bouger un tenseur sur une surface peut dévier ses directions si aucune précaution n'est prise. Ici, nous proposons une manière sûre de les déplacer sans modifier leurs directions principales. Elle est basée sur la notion de **transport parallèle** en géométrie différentielle, et contribue à accélérer la convergence du maillage en termes d'approximation de la surface et d'interpolation de la solution numérique.

In fine, la contrainte de localité induite par le hardware est bien respectée tout en préservant l'efficience des noyaux. D'une part, nos quatre noyaux n'impliquent qu'un voisinage restreint et statique pour le rééchantillonnage de la surface, la régularisation des mailles et la relaxation des degrés (figure 3.2). D'autre part, leur efficience est garantie car ils s'appuient sur trois primitives géométriques avancées qui leur permettent de converger rapidement en erreur et qualité. Enfin, notons qu'ils ne nécessitent aucun support géométrique<sup>4</sup> et préservent les particularités de la géométrie<sup>5</sup>. Pour finir, une partie des travaux a été publiée dans un acte de conférence avec comité de lecture [RL18], et présentée oralement à un congrès [LR18].

**[RL18]** Accurate manycore-accelerated manifold surface remesh kernels.

Hoby Rakotoarivelo and Franck Ledoux.

27<sup>th</sup> International Meshing Roundtable, 2018, Albuquerque NM, USA.

**[LR18]** Parallel surface mesh adaptation for manycore architectures.

Franck Ledoux and Hoby Rakotoarivelo. Communication orale à MeshTrends  
13<sup>th</sup> World Congress in Computational Mechanics, 2018, New-York NY, USA.

## 3.2 DESIGN DES NOYAUX

### 3.2.1 Hypothèses et optimalité

HYPOTHÈSES. Ici, la surface fournie en entrée est connue uniquement par le biais d'une triangulation conforme au sens de la définition 4. Ainsi la surface triangulée est une reconstruction linéaire par

<sup>4</sup>Comme un moteur de CAO ou un maillage de fond pour les requêtes relatives à la surface par exemple

<sup>5</sup>telles que les courbes saillantes et "coins" ou points singuliers

morceaux de la surface idéale, et on suppose que :

- la surface est une variété lisse par morceaux (voir définitions 6 et 8). Ainsi tout point de la surface admet un unique plan tangent, et est indéniablement différentiable presque partout sauf sur les courbes saillantes ("ridges") ou les "coins". Cela permet de calculer des quantités différentielles utiles pour définir une carte de tailles basée sur l'erreur de la surface ou juste pour reconstruire localement la surface.
- la surface est une variété fermée, orientable et sans bords (définitions 9 et 11). Ainsi, elle représente une frontière d'un domaine volumique, et peut être munie d'un champ de vecteurs normaux de sorte à distinguer l'intérieur et l'extérieur de ce domaine.
- la triangulation est interpolante (et non approximante, voir définition 5). Ainsi ses points sont exactement sur la surface idéale, et elle tend vers cette surface quand le pas de discrétisation<sup>6</sup> tend vers zéro.

OPTIMALITÉ. La surface initiale peut être sous ou sur-échantillonnée<sup>7</sup>. Étant donné un budget de points  $n_{\max}$ , le but consiste à effectuer des opérations locales sur la triangulation initiale, et donc de produire une séquence de triangulations qui converge vers une discrétisation "optimale". Elle est :

- *proche* de la surface *idéale*, au sens où l'erreur estimée de la surface est minimale pour le nombre de points spécifié. Notons que cet erreur décroît strictement dans le cas d'un raffinement. Ici nous utilisons une norme  $L^p$  pour l'évaluation de l'erreur, cela permet d'utiliser des distances différentes et donc d'ajuster la sensibilité de l'estimateur<sup>8</sup>. Enfin, notons qu'on cherche à minimiser l'erreur compte-tenu du nombre de points et pas l'inverse<sup>9</sup>. En fait, contrôler le nombre de points est souvent préférable en calcul numérique : c'est pratique quand le numéricien veut spécifier le nombre exact de points pour une étude de convergence par exemple.
- *bien échantillonnée* c'est-à-dire constituée de mailles de *bonne qualité*<sup>10</sup>. Par souci de simplicité, nous décidons de décrire la qualité d'une maille par le *radii-ratio* (voir table 2.1, page 43). Notons toutefois que les grandeurs géométriques<sup>11</sup> sont calculés dans l'espace métrique induit par la carte de tailles définie sur les points du maillage. Pour une maille  $K$ , sa qualité est définie par :

$$q[K] = \frac{R}{2r}, \begin{cases} R : \text{rayon du cercle circonscrit de } K \\ r : \text{rayon du cercle inscrit de } K \end{cases} \quad (3.1)$$

Ainsi elle est de bonne qualité si  $q[K] \approx 1$  et dégénérée si  $q[K] \approx 0$ .

- *régulière* dans le sens où chaque maille est proche des plans tangents de ses sommets d'une part et que les tailles d'arêtes varient graduellement d'autre part. La première contrainte permet d'éviter une discrétisation certes proche de la surface mais toute crénélée, tandis que la seconde est nécessaire pour un calcul<sup>12</sup> plus stable de quantités numériques sur les points. Concrètement, nous fixons un seuil angulaire  $\theta_{\max}$  sur la déviation des normales aux points et aux mailles d'une part, ainsi qu'un seuil sur le rapport de tailles d'arêtes incidentes d'autre part.

### 3.2.2 Pré-traitement

Rappelons que nous ne disposons que de données minimales relatives à la description de la surface, à savoir un ensemble de points et de mailles sans aucune autre donnée géométrique, tels que les modèles fournis au format STL (STereo-Lithography). Ainsi une étape de pré-traitement incluant l'extraction de données additionnelles est nécessaire. Les étapes afférentes sont résumées à la figure 3.3.

<sup>6</sup>Il est représenté par le diamètre  $h$  du maillage  $T_h$  qui correspond à la longueur de sa plus grande arête.

<sup>7</sup>Comme les modèles géométriques issus d'une CAO ou de capteurs 3D.

<sup>8</sup>En norme  $L^2$ , elle correspond à la distance euclidienne entre les points de  $\mathcal{T}$  et ceux de  $\Gamma$ . En norme  $L^\infty$ , il s'agit de la distance de Hausdorff qui représente l'écart maximal entre  $\mathcal{T}$  et  $\Gamma$ .

<sup>9</sup>On ne cherche pas à minimiser  $n$  étant donné un seuil d'erreur  $\varepsilon_{\max}$  – contrairement aux méthodes usuelles en adaptation de maillages [78].

<sup>10</sup>Notons que la forme et l'alignement d'un triangle sont relatives à la variation de la solution numérique ou de la surface, il ne s'agit pas juste d'avoir un triangle équilatéral.

<sup>11</sup>Comme les distances, angles etc.

<sup>12</sup>Pour l'interpolation ou la quadrature lors d'une intégration numérique par exemple

### 3.2.2.1 Extraction des ridges

EN BREF. Avant tout autre traitement, il nous fait extraire les caractéristiques importantes de la surface que l'on souhaite préserver. Il s'agit d'identifier les entités de discontinuité<sup>13</sup> qui vont nécessiter un traitement particulier tout au long du remaillage. Dans notre cas, on distingue :

- les "ridges" : ce sont les courbes caractéristiques délimitant deux portions de la surface qui s'intersectent avec un angle aigu (crêtes ou vallées). Pour les modèles issus d'une CAO, elles peuvent être identifiées en repérant simplement les paires de mailles dont l'angle dièdre excède un certain seuil fixé. Pour les variétés plus générales, leur extraction n'est pas triviale, et nécessite l'estimation des extrema des courbures principales sur les points de la surface [137–139].
- les "coins" : ce sont les points saillants de la surface qui doivent être préservés. Ils sont identifiés comme étant l'intersection d'au moins trois ridges.

Les autres entités sont dites *ordinaires* et correspondent aux régions où la surface est localement lisse.

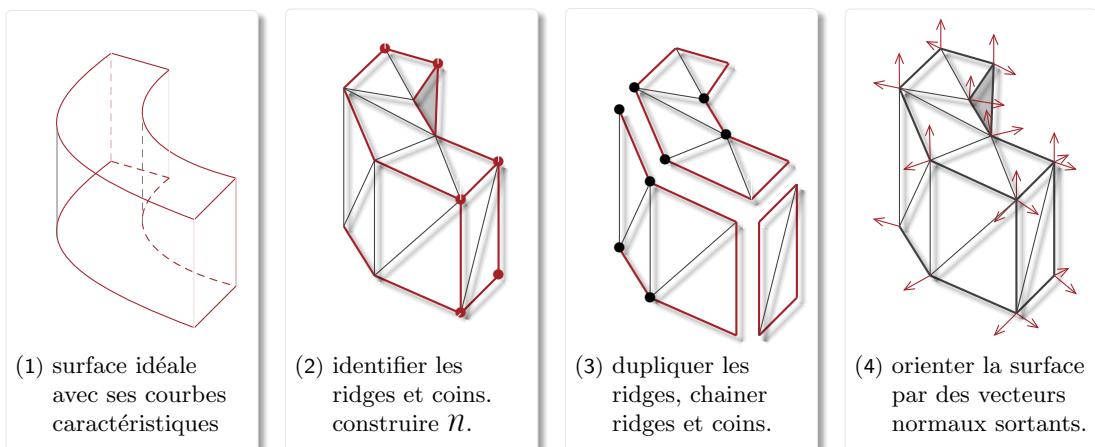


Figure 3.3: Étapes de pré-traitement.

### 3.2.2.2 Représentation de la topologie

Une fois ses entités classifiées, il nous faut construire la topologie de  $\mathcal{T}_h$ , i.e la connectivité des points et des mailles. Il y a plusieurs manières de représenter cette topologie (voir chapitre précédent). Dans notre cas, on veut un modèle de représentation qui :

1. est *local* afin de restreindre les mailles impactées lors d'une mise à jour.
2. est *minimal* afin de simplifier les mises à jour.
3. minimise les *indirections* afin de réduire les défauts de cache lors des requêtes.

GRAPHE D'INCIDENCE. Les surfaces triangulées sont usuellement représentées par une structure orientée telle que les *halfedges* ou les *cartes combinatoires* [140, 109]. Ainsi la topologie est portée exclusivement par les arêtes qui sont dupliquées. L'avantage est double. D'une part, leur mises à jour restent *locales* à la maille ou à la boule d'un point en cours de modification. D'autre part, une structure orientée permet la vérification d'invariants au sein de chaque noyau avant et après modification. Néanmoins elle comporte deux inconvénients :

- un surcoût mémoire important lié au stockage des demi-arêtes en  $O(6n)$ , ce qui ne permet pas de passer à l'échelle en vue d'un calcul intensif,
- un nombre important d'indirections lors des requêtes de voisinage. Ici reconstruire la boule d'un point implique  $2r$  indirections, avec  $r$  le degré du point : cela peut induire autant de défauts de cache si les données associées ne sont pas voisines en mémoire.

Une autre représentation usuelle consiste à extraire le graphe dual de la triangulation, i.e le voisinage des mailles. Bien que simple à mettre à jour, elle implique presque autant d'indirections que les

<sup>13</sup>Ils sont indiqués par  $\Delta$  RIDGES dans tout le manuscrit.

halfedges sans leurs avantages. Devant ce constat, nous décidons de stocker uniquement les mailles incidentes à  $p$ . Pour un point donné, on peut néanmoins retrouver ses points voisins avec un seul niveau d'indirection. Ainsi la topologie est représentée sous forme d'un graphe biparti  $(P, M, \mathcal{N})$  avec :

- $P = \{p_i\}_{i \in V}$  : l'ensemble des points,
- $M = \{K_j\}_{j \in K}$  : l'ensemble des mailles, qui sont en fait des triplets d'indices de points.
- $n : V \rightarrow \mathcal{P}(K)$  donne pour chaque point les mailles qui lui sont incidentes<sup>14</sup>.

**△ RIDGES.** Si la surface comporte des discontinuités alors le traitement est assez complexe, puisqu'on a besoin d'une représentation explicite des arêtes vives. On aurait pu juste marquer ou stocker uniquement leurs sommets (ou ridges), toutefois toute paire de ridges ne correspond pas forcément à une arête vive (voir figure 3.3). Pour ne pas devoir utiliser une représentation centrée arête, on décide de stocker l'adjacence des coins et ridges dans un graphe  $(S, R, \beta)$ .

- $S$  : l'ensemble des "coins",
- $R$  : l'ensemble des ridges,
- $\beta : R \rightarrow \mathcal{P}(S \cup R)$  permet de retrouver les arêtes vives incidentes à un ridge ou coin.

Pour rester cohérent, on a besoin de relier les indices des ridges et coins à leurs coordonnées. Pour cela on maintient un mapping bijectif<sup>15</sup>  $\nu : E \rightarrow R \cup S$ , où  $E \subset V$  est formé par les indices des ridges.

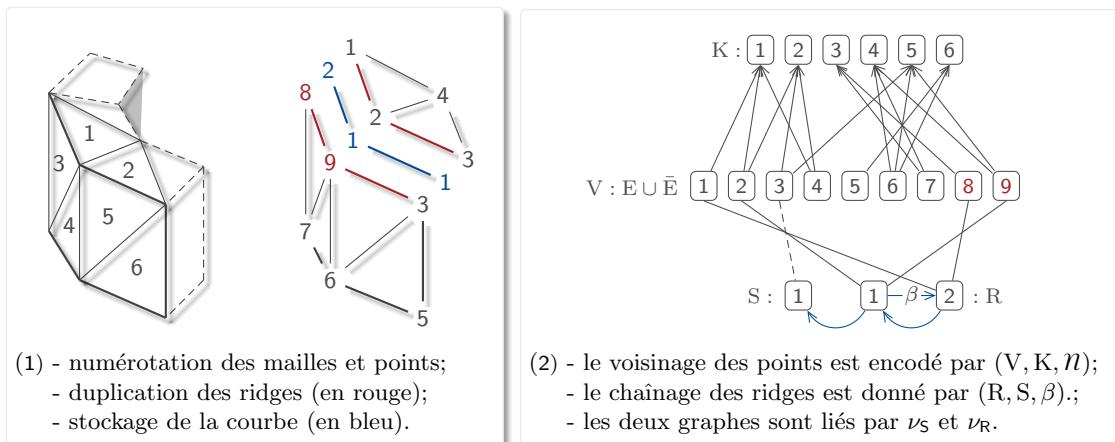


Figure 3.4: Construction de la topologie.

**DUPLICATION.** En fait, il est crucial d'avoir une représentation des arêtes vives qui soit indépendante d'un voisinage pour la parallélisation des noyaux. En réalité, les traitements sur un ridge  $p$  délimitant deux régions  $\Gamma_1$  et  $\Gamma_2$  implique souvent des données propres :

- à son sous-voisinage appartenant à une région  $\Gamma_i$ ,
- ou bien aux sommets des deux arêtes vives incidentes à  $p$ .

Devant ce constat et du fait qu'on ne puisse extraire un sous-voisinage de manière directe, on décide de dupliquer les ridges (figure 3.41). Ici l'avantage est double. Pour chaque ridge, cela permet d'attacher des données relatives à chaque région sans avoir à les re-calculer à la volée à chaque fois. De plus, cela permet aux noyaux de traiter simultanément ces deux régions. Concrètement, l'ensemble de points  $V$  est enrichi par les points dupliqués  $\bar{E}$ , et la topologie  $\mathcal{N}$  est modifiée afin de discriminer les deux sous-voisinages de chaque ridge. Notons que  $R$  et  $S$  restent inchangés, mais le mapping  $\nu$  est scindé en deux  $\nu_R : E \times \bar{E} \rightarrow R$  et  $\nu_S : E \rightarrow S$  (figure 3.42). Pour ce qui est du surcoût-mémoire, il reste minime dans la mesure où le nombre de ridges est faible comparé au nombre total de points<sup>16</sup>. Enfin chaque graphe est stocké en mémoire sous forme d'un tableau de listes d'incidences<sup>17</sup>

<sup>14</sup>Les  $V, K \subset \mathbb{N}$  sont respectivement les espaces d'indexation des points et mailles.

<sup>15</sup>Concrètement, on stocke deux tables  $\nu$  et  $\nu^{-1}$  qui seront mises à jour par les noyaux.

<sup>16</sup>Du moins si le maillage est bien échantillonné, ce qui est le but du remaillage.

<sup>17</sup>Notons qu'il est moins compact que la structure usuelle CSR dédié au stockage de graphes et matrices creuses [208, 209]. Néanmoins elle est plus flexible pour les mises à jour, et les deux sont équivalentes en terme de compacté quand

### 3.2.2.3 Orientation de la surface

Enfin, la dernière étape consiste à munir la surface triangulée d'une orientation consistante<sup>18</sup>. Comme la surface est une frontière d'un domaine volumique, l'idée est de distinguer l'intérieur et l'extérieur de ce domaine. Pour cela, il nous faut extraire un champ de vecteurs normaux qui pointe vers son extérieur. Pour chaque point, le vecteur normal associé est orthogonal à son plan tangent.

PRINCIPE. D'abord il nous faut orienter les mailles<sup>19</sup>, ensuite on calcule la normale à un point à partir des mailles qui lui sont incidentes. Pour orienter les mailles, l'idée est d'ordonner ses trois sommets selon un sens fixé au préalable. Pour cela,

- on commence par orienter une maille arbitraire  $K$  (ou graine) en fixant une normale sortante à  $K$ , et telle que ses sommets soient réordonnés dans le sens trigonométrique dans ce repère local ainsi formé. Cette normale s'obtient par un produit vectoriel des composantes de la matrice jacobienne  $J_K \in \mathbb{R}^{2 \times 3}$  associée à  $K$ .
- on se propage ensuite sur la surface grâce un parcours en largeur à partir de  $K$ .

Enfin, il reste à orienter les points. Ici, un point régulier admet une unique normale sortante car la surface est localement lisse (et donc dérivable) en ce point. Dans notre cas, la normale à un point correspond à une moyenne pondérée des normales unitaires des mailles qui sont incidentes à ce point<sup>20</sup>. Il y a plusieurs choix possibles pour les poids (angles incidents, aire des mailles). Nous retenons le critère d'angle car c'est assez stable : il ne dépend pas de la taille des mailles [141, 142].

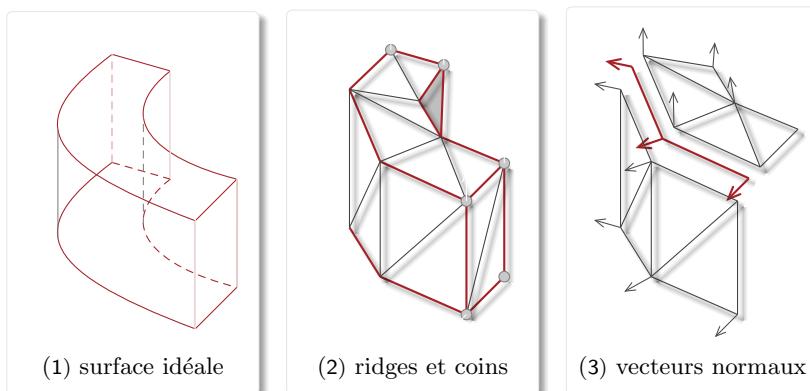


Figure 3.5: Orientation de la surface par extraction des normales.

$\triangle$  RIDGE. Si le point  $p$  est un ridge ou un coin, alors on perd l'unicité de la normale sortante. En fait le traitement diffère pour les deux cas :

- ridge : ici,  $p$  appartient à une unique courbe délimitant deux régions distinctes de la surface. Ainsi il admet une normale sortante par région, celle-ci est calculée comme dans le cas d'un point régulier, mais sur uniquement le sous-voisinage de  $p$  dans cette région. Par ailleurs,  $p$  est un point de contrôle d'une courbe caractéristique  $\gamma$  à préserver. Ainsi nous décidons de calculer et stocker la normale relative à la dérivée seconde de  $\gamma$  sur  $p$ . Cela permet de reconstruire  $\gamma$  indépendamment de la région qu'on considère (figure 3.5).

la taille des listes est figée, notamment pour le chainage des ridges.

<sup>18</sup>Ici le fait que la surface soit une variété orientable est important.

<sup>19</sup>Notons que le fait d'orienter les mailles permet de détecter les inconsistances topologiques (comme les trous ou replis) lorsqu'on applique un noyau.

<sup>20</sup>En fait, il y a une autre manière de calculer la normale en un point : le *tensor voting*. Dans ce cas, cette normale s'obtient en minimisant l'erreur d'une surface quadrique définie sur un point  $p$ , puis en faisant une décomposition spectrale de la matrice  $M$  définissant la quadrique avec  $A_p = -B$ .  $n(p)$  correspond au vecteur propre associé à la plus grande valeur propre de  $M$ . En fait la réelle différence entre ces deux approches réside dans l'estimation des normales aux lignes de  $C^1$ -discontinuité ou quand la triangulation est trop bruitée. Mais compte-tenu de nos hypothèses sur la surface triangulée en entrée, la manière de calculer chaque normale importe peu in fine.

- coin : ici, la surface n'est pas dérivable et le point n'admet pas un unique plan tangent. Il nous faut quand même définir une normale pour ces points là. Ainsi cette normale n'est pas stockée mais va être calculée à la volée selon la région de la surface que l'on considère.

Maintenant que les points sont munis d'une orientation consistante, on peut à présent reconstruire localement la surface idéale, ou estimer des quantités relatives à sa variation (gradient, courbure).

### 3.2.3 Reconstruction et projection

Comme énoncé auparavant, nous devons trouver une manière de placer correctement les points sur la surface idéale lors du raffinement, de la simplification ou du lissage. En fait, si le point est déjà sur la triangulation, alors il peut être directement projeté sur cette surface par le biais d'une paramétrisation locale. Le vrai problème réside dans le cas où ledit point n'est pas sur la triangulation. Ici nous montrons d'abord comment nous reconstruisons localement cette surface idéale. Ensuite, nous montrons comment nous projetons le point de manière précise sur cette surface idéale.

BUT. En fait, la surface idéale  $\Gamma$  n'est connue que sur les points de la triangulation<sup>21</sup>, et doit être reconstruite localement. Cela peut être fait de plusieurs manières (voir page 49), néanmoins il est difficile de concilier à la fois précision et régularité. Par exemple,

- elle peut être approchée dans le voisinage d'un point à l'aide d'une surface quadrique obtenue par une méthode des moindres carrés [ref]. Comme  $\Gamma$  est approchée par une surface lisse, alors on a bien la régularité. Néanmoins elle ne fournit qu'une reconstruction quadratique de  $\Gamma$ .
- elle peut être retrouvée par interpolation cubique grâce aux PN-triangles<sup>22</sup> [143]. C'est ce qui est souvent utilisé par les remailleur et shaders tels que MMGS, GMSH ou le moteur de tessellation DIRECT-X11 [119, 132, 40]. Bien qu'ils soient compacts et efficaces<sup>23</sup>, ils ne garantissent pas l'unicité des plans tangents des points sur les frontières des patchs. Pour cela, il faudrait plus de degrés de liberté, et donc plus de points de contrôle internes pour chaque maille<sup>24</sup>.

Ici, nous visons à concilier les deux contraintes (précision et régularité). En fait, nous utilisons une paramétrisation locale par maille avec une continuité  $G^1$  complète comme sur la figure 3.6. Un patch spline quartique est calculé sur chaque maille à partir des normales en ses sommets, de sorte que l'unicité du plan tangent de chaque point soit garantie. La surface est ensuite reconstruite en connectant les patchs entre eux. Pour cela, nous construisons un PATCH DE GREGORY avec un BLENDING des points twists comme dans [129]. Néanmoins nous traitons différemment les ridges, coins et la construction du ruban tangent<sup>25</sup>.

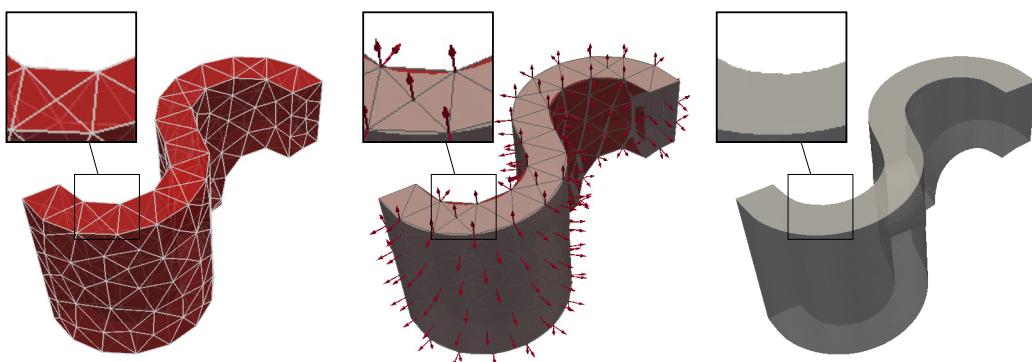


Figure 3.6: Reconstruction complète de la surface idéale à partir de la triangulation.

<sup>21</sup>Puisqu'il s'agit d'une triangulation interpolante.

<sup>22</sup>Ils visent à fournir une paramétrisation cubique de chaque maille. Plus précisément, il s'agit de triangles de Béziers dont les 10 points de contrôle ne dépendent que des points et des normales de K

<sup>23</sup>Ils nécessitent peu de calcul et peuvent être stockées de manière efficiente en cache.

<sup>24</sup>En effet il a été montré qu'il faut au minimum des triangles quartiques pour reconstruire une surface  $G^1$  [144].

<sup>25</sup>Un ruban tangent est un champ de vecteurs tangents sur une courbe de la surface.

### 3.2.3.1 Patch spline par maille

Ici, on vise à construire un patch quartique qui permet de retrouver la surface sur chaque maille<sup>26</sup> en ne connaissant que les normales en ses sommets (figure 3.7). Elle est construite de manière à ce que la surface reconstruite soit la plus régulière possible quand on connecte les patchs. Pour cela, on souhaite que chaque frontière d'un patch coïncide avec les géodésiques<sup>27</sup> de la surface passant par ses sommets. De plus, il faudrait que le plan tangent de tout point  $p$  aux sommets ou à la jonction des patchs soit défini de manière unique. Ainsi tous les vecteurs tangents à  $p$  doivent être coplanaires.

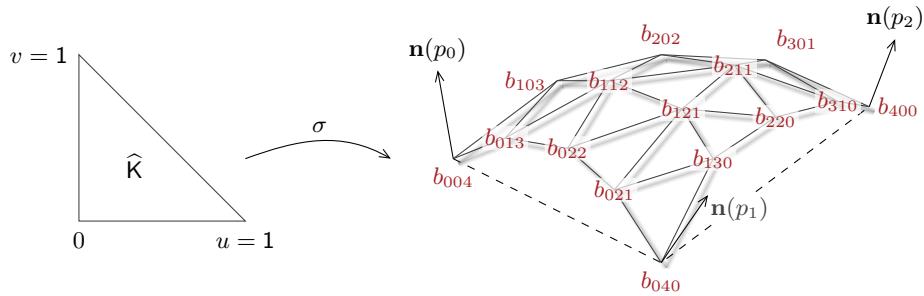


Figure 3.7: Reconstruction locale à une maille par un patch quartique.

CONSTRUCTION. Le but est donc de choisir correctement les points de contrôle afin de s'assurer que les vecteurs tangents à un point à la frontière d'un patch soient coplanaires. Ici, les points de contrôle des trois sommets du patch coïncident<sup>28</sup> avec ceux de la maille :

$$(b_{0,0,4} \quad b_{0,4,0} \quad b_{4,0,0}) = (p_0 \quad p_1 \quad p_2) \quad (3.2)$$

Pour les points de contrôle  $(b_{0,j,k})$ ,  $(b_{i,0,k})$  et  $(b_{i,j,0})$  internes à chaque courbe frontière d'un patch, on suit l'approche de [129]. En fait, on veut que les dérivées secondes de chaque courbe  $\gamma$  aux sommets de  $K$  coïncident avec les normales spécifiées en ces sommets, i.e.  $\langle \frac{d\gamma}{ds}(0), \frac{d^2\gamma}{ds^2}(0) \rangle = \langle \frac{d\gamma}{ds}(0), \mathbf{n}[p_i] \rangle = 0$ , où  $\langle \cdot, \cdot \rangle$  est le produit scalaire usuel de  $\mathbb{R}^3$ . En effet cette condition est nécessaire et suffisante pour la coplanarité des vecteurs tangents aux courbes incidentes à chaque  $p_i$ . Concrètement, l'idée est de construire des B-splines cubiques dans un premier temps, puis de procéder à une élévation de degré. En imposant que ses dérivées premières et secondes sur  $p_i$  et  $p_{i+1}$  soient en plus orthogonales<sup>29</sup>, on obtient une répartition unique des points de contrôle cubiques  $(c_{i,j})_{j=0}^3$  associés à  $\gamma_i$ . Elle s'écrit :

$$\forall \gamma_i \in \Gamma_K, \begin{bmatrix} c_{i,0} \\ c_{i,1} \\ c_{i,2} \\ c_{i,3} \end{bmatrix} = \begin{bmatrix} p_i \\ c_{i,0} + d_i(6\mathbf{t}_i - 2\rho_i \mathbf{n}_i + \mu_i \mathbf{n}_{i+1}) \\ c_{i,3} - d_i(6\mathbf{t}_i + \rho_i \mathbf{n}_i - 2\mu_i \mathbf{n}_{i+1}) \\ p_{i+1} \end{bmatrix} \quad (3.3)$$

$$\text{avec } \begin{bmatrix} \rho_i \\ \mu_i \end{bmatrix} = \frac{6}{4 - a_i^2} \begin{bmatrix} 2a_{i,0} + a_i a_{i,1} \\ 2a_{i,1} + a_i a_{i,0} \end{bmatrix}, \text{ et } \begin{cases} d_i = \frac{1}{18} \|p_{i+1} - p_i\| \\ a_i = \langle \mathbf{n}_i, \mathbf{n}_{i+1} \rangle \\ a_{i,0} = \langle \mathbf{n}_i, \mathbf{t}_i \rangle \\ a_{i,1} = \langle \mathbf{n}_{i+1}, \mathbf{t}_i \rangle \end{cases} \quad (3.4)$$

<sup>26</sup>Ici, chaque maille représente une région lisse de la surface, mais peut contenir des ridges ou coins.

<sup>27</sup>Une géodésique entre deux points  $p$  et  $q$  est le plus court chemin entre  $p$  et  $q$  sur la surface. En fait, ce sont les courbes passant par  $p$  et  $q$  mais de longueur et variation minimales (il peut y en avoir plusieurs).

<sup>28</sup>Car la triangulation est interpolante.

<sup>29</sup>Les indices sont modulo 3 bien entendu.

Enfin, les points de contrôle quartiques  $(q_{i,j})_{j=0}^4$  associés à chaque frontière  $\gamma_i$  s'obtiennent par simple élévation de degré [145] des  $(c_{i,j})_{j=0}^3$  :

$$\begin{aligned} \forall \gamma_i \in \Gamma_K, (q_{i,j})_{j=0}^4 &= (\frac{j}{4}c_{i,j-1} + \frac{4-j}{4}c_{i,j})_{j=1}^3 \\ \text{et on a : } (b_{0,j,k})_{j+k=4} &= (q_{0,j})_{j=1}^3 \\ (b_{i,0,k})_{i+k=4} &= (q_{1,j})_{j=1}^3 \\ (b_{i,j,0})_{i+j=4} &= (q_{2,j})_{j=1}^3 \end{aligned} \quad (3.5)$$

**RIDGES.** Dans (3.3) et (3.4), le terme  $\mathbf{t}_i$  correspond à un vecteur tangent à  $p_i$  relatif à la corde de  $\gamma_i$ , et  $\mathbf{n}_i$  est un vecteur normal à  $p_i$ . Afin de tenir compte des ridges et coins, ils doivent être choisis avec précaution. Ici, ils correspondent à :

$$\mathbf{n}_i = \begin{cases} \mathbf{n}(p_i) & \text{si } p_i \text{ est régulier.} \\ \mathbf{n}(K) & \text{si } p_i \text{ est un coin.} \\ \mathbf{n}_j(p_i) & \text{si } p_i \text{ est un ridge délimitant } \Gamma_1 \cap \Gamma_2 \text{ et } K \text{ appartient à } \Gamma_{j=1,2}. \end{cases} \quad (3.6)$$

$$\mathbf{t}_i = \begin{cases} \frac{p_{i+1} - p_i}{\|p_{i+1} - p_i\|} & \text{si } p_i \text{ est régulier ou un coin.} \\ \frac{\mathbf{n}_{i,1} \times \mathbf{n}_{i,2}}{\|\mathbf{n}_{i,1} \times \mathbf{n}_{i,2}\|} & \text{si } p_i \text{ est un ridge de } \Gamma_1 \cap \Gamma_2, \text{ et } \mathbf{n}_{i,j} = \mathbf{n}_j(p_i). \end{cases} \quad (3.7)$$

À ce point, on dispose d'une reconstruction des courbes frontières des mailles telle que ses dérivées secondes soient alignées avec les normales spécifiées en ses sommets. Cela nous assure qu'en tout point où plusieurs courbes frontières se rejoignent, leurs vecteurs tangents relatifs à leurs sommets sont coplanaires. Maintenant, il nous reste à définir les trois points de contrôle internes  $b_{1,1,2}$ ,  $b_{1,2,1}$  et  $b_{2,1,1}$  du triangle quartique.

### 3.2.3.2 Blending pour la continuité $G^1$

**PRINCIPE.** Nous avons résolu la contrainte d'unicité du plan tangent sur chaque sommet des mailles. À présent, le problème est de trouver une configuration des points twists, de sorte que tous les vecteurs tangents  $\{\mathbf{t}_{i,j}\}_{j \in \mathbb{N}}$  en un point à la jonction de patchs soient mutuellement coplanaires. Le problème de trouver ces points twists s'appelle problème de **consistance au sommets** dans la littérature<sup>30</sup>. Nous avons opté pour un blending basé sur un PATCH DE GREGORY [146]. L'avantage est qu'il est simple à implémenter et pas trop coûteux<sup>31</sup>. De manière concrète, on construit d'abord deux points twists  $\mathbf{f}_i^+$  et  $\mathbf{f}_i^-$  pour chaque sommet  $p_i$  de  $K$  tel qu'illustré à la figure 3.81. Ensuite les trois points twists finaux sont obtenus par interpolation des six points twists intermédiaires. En notant  $u, v$  et  $w$  les coordonnées barycentriques du point à projeter dans  $K$ , on a :

$$\forall u, v, w \in ]0, 1[, (b_{1,1,2} \ b_{1,2,1} \ b_{2,1,1}) = \left( \frac{u\mathbf{f}_0^+ + v\mathbf{f}_0^-}{u+v} \quad \frac{w\mathbf{f}_2^+ + u\mathbf{f}_2^-}{w+u} \quad \frac{v\mathbf{f}_1^+ + w\mathbf{f}_1^-}{v+w} \right) \quad (3.8)$$

Notons que le cas où l'un des  $u, v$  ou  $w$  s'annule n'est pas un problème, puisque le point est sur une courbe frontière dans ce cas, et il peut directement être projeté (voir annexe en page 133).

<sup>30</sup>En fait, il est usuellement résolu par l'une des trois stratégies suivantes [110, 111] :

- **splitting** : ici, chaque patch est scindé en trois patchs  $\Gamma_{i,K}$  selon un motif particulier. Les points twists de chaque  $\Gamma_{i,K}$  peuvent être calculés par une méthode basée sur la construction d'un ruban tangent le long des frontières.
- **boundary conditioning** : ici, la contrainte est directement intégrée lors de la construction des courbes frontières de sorte qu'elles soient  $C^2$ -compatibles. Les points twists sont ensuite obtenus par résolution d'un système d'équations incluant ces contraintes.
- **blending** : on construit un ruban tangent le long de chaque frontière dans un premier temps. Ensuite, on génère trois patchs intermédiaires  $\Gamma_{i,K}$  interpolant une partie de chaque ruban. Les points twists du patch final s'obtiennent par combinaison convexe de ceux de  $\Gamma_{i,K}$ .

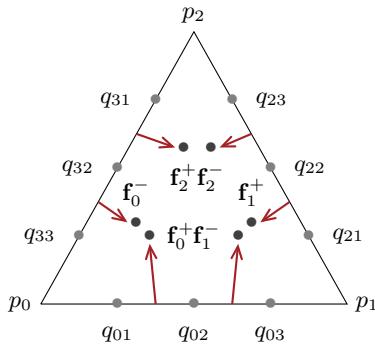
<sup>31</sup>En effet, on ne crée pas réellement les patchs intermédiaires, et on ne résout pas de système d'équations directement, comparé aux deux autres stratégies.

DÉTAILS. En reprenant les notations de (3.5), les points twists  $\mathbf{f}_i^-$  et  $\mathbf{f}_i^+$  sont calculés comme suit :

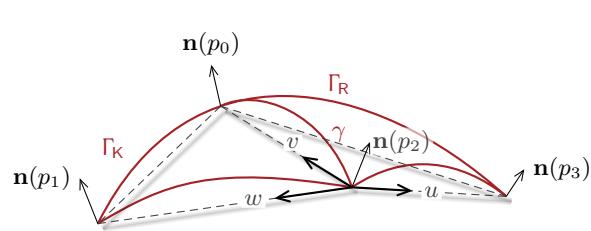
$$(\mathbf{f}_i^+, \mathbf{f}_i^-) = \left[ \frac{1}{2}(q_{i,1} + q_{i,2}) + \mathbf{t}_{i,1}, \frac{1}{2}(q_{j,2} + q_{j,3}) + \mathbf{t}_{j,2} \right], \quad \begin{cases} 0 \leq j \leq 2 \\ i = j + 1 \text{ mod } 3 \end{cases}. \quad (3.9)$$

Dans (3.9), les vecteurs tangents  $\{\mathbf{t}_{i,j}\}_{j=0}^3$  coïncident avec les dérivées de la surface sur chaque courbe frontière d'une maille. Pour avoir la continuité, ils doivent être coplanaires. En d'autres termes, ils doivent définir un unique ruban tangent pour les deux mailles incidente à une courbe  $\gamma$ . Pour cela, ils sont calculés de sorte que les vecteurs tangents et binormaux des points de contrôle de  $\gamma_i$  appartiennent à ce ruban tangent [147]. Notons  $\gamma$  la courbe commune à deux patchs  $\Gamma_K$  et  $\Gamma_R$  comme sur la figure 3.8. Le ruban tangent est déterminé en résolvant le système suivant :

$$\begin{cases} \frac{\partial \Gamma_K}{\partial u}(0, t) = \lambda_K[t]\gamma[t] + \frac{1}{4}\mu_K[t]\frac{d\gamma}{dt}(t) \\ \frac{\partial \Gamma_R}{\partial w}(0, t) = \lambda_R[t]\gamma[t] + \frac{1}{4}\mu_R[t]\frac{d\gamma}{dt}(t) \end{cases}, \quad \forall t \in [0, 1]. \quad (3.10)$$



(1) points twists du patch de GREGORY



(2) courbe entre deux patchs, et repère local.

Figure 3.8: Points twists et vecteurs impliqués dans la construction du ruban tangent.

Pour chaque courbe  $\gamma_i$ , les  $\{\mathbf{t}_{i,j}\}$  s'obtiennent en exprimant leur dérivées dans la base de BÉZIERS mais sur les milieux  $\{m_{i,j}\}_{j=0}^3$  des segments de contrôle de  $\gamma$  cette fois, comme dans (3.11).

$$\frac{\partial \Gamma_K}{\partial u}(0, t) = \lambda_K[t] \sum_{j=0}^2 \mathbf{B}_j^2[t] m_{i,j} + \mu_K[t] \sum_{j=0}^2 \mathbf{B}_j^2[t] (m_{i,j+1} - m_{i,j}). \quad (3.11)$$

Ici, chaque  $\mathbf{B}_i^n(t)$  est un polynôme de BERNSTEIN<sup>32</sup> de degré  $n$ . Notons que les  $\{\mathbf{t}_{i,j}\}$  peuvent être calculés individuellement sur chaque maille, puisque  $\frac{\partial \Gamma_K}{\partial v}(0, v) = \frac{d\gamma}{ds}(t)$  ne dépend que des points de contrôle de la courbe en question. Enfin les  $\lambda_K(t)$  et  $\mu_K(t)$  décrits en (3.10) sont des polynômes linéaires qui décrivent l'ajustement des dérivées de la courbe, de sorte qu'ils appartiennent au ruban tangent<sup>33</sup>. Leurs coefficients peuvent être déterminés en évaluant les dérivées transverses aux sommets de la courbe, c'est-à-dire en  $v = 0$  et  $v = 1$ . En effet  $\mathbf{t}_{i,0}$  et  $\mathbf{t}_{i,3}$  sont connus en ces points puisqu'ils appartiennent aux plans tangents des sommets de  $K$ .

### 3.2.3.3 Notre opérateur de projection

Grâce à ces points twists, nous disposons d'une reconstruction complète de la surface idéale sur la triangulation (figure 3.6), et qui respecte l'unicité des plans tangents en tout point des patchs : ainsi la surface reconstruite est bien une variété (définition 6). Plus précisément, nous avons construit un opérateur permettant de projeter tout point de la triangulation sur la surface idéale. Pour cela, le point  $p$  situé sur la maille  $K$  est d'abord paramétrisé en calculant ses coordonnées barycentriques  $(u, v)$  dans  $K$ . Ensuite, selon les valeurs de  $u$ ,  $v$  et  $w = 1 - u - v$  on identifie deux cas de figure :

<sup>32</sup>En fait, le polynôme de BERNSTEIN de degré  $n$  est :  $\mathbf{B}_j^n(t) = \frac{n!}{j!(n-j)!}(1-t)^n t^j$ ,  $\forall j, n \in \mathbb{N}$ , cf. [145].

<sup>33</sup>En fait,  $(\lambda_K, \mu_K)(t) = (\sum_{j=0}^1 \mathbf{B}_j^1(t)\lambda_{i,j}, \sum_{j=0}^1 \mathbf{B}_j^1(t)\mu_{i,j})$ ,  $\forall t \in [0, 1]$ .

Notons qu'en injectant chaque terme de cette équation dans (3.11), on a bien une interpolation cubique des dérivées transverses.

- si  $u = 0$  ou  $v = 0$  ou  $w = 0$  alors le projeté de  $p$  est déterminé par l'équation A.2.
- sinon on calcule les trois points twists selon l'équation 3.8, et le projeté de  $p$  s'obtient ensuite par l'équation A.1.

PROBLÈME. In fine, le projeté d'un point sur la surface idéale s'obtient de manière directe tant que son plan tangent est supporté par une maille. En pratique, nous sommes parfois amenés à projeter un point qui n'est pas directement sur la triangulation, et qui n'est donc pas directement paramétrable. Durant le lissage par exemple, le projeté  $q$  d'un point  $p$  doit être calculé de manière à ce que la longueur du segment courbe géodésique  $\gamma$  sous-tendue par  $[pq]$  dirigé par un vecteur déplacement  $t$  doit être égale à  $\|t\|$  (sec. 3.2.4.4). Autrement dit, il faudrait disposer d'une paramétrisation de  $\gamma$  par la longueur d'arc. Dans le cadre continu, cet opérateur existe : l'application exponentielle<sup>34</sup> (définition 28).

**Définition 28** (APPLICATION EXPONENTIELLE). Soit  $p$  un point d'une variété  $\Gamma$  et  $R$  une région de son plan tangent  $T_p\Gamma$ . Son application exponentielle notée  $\exp_p : R \rightarrow \Gamma$  associe à tout vecteur tangent  $t$  de  $R$  le segment courbe géodésique  $\gamma$  d'origine  $p$ , de vitesse initiale  $\frac{d\gamma}{dt}(0) = t$  et de longueur  $\ell(\gamma) = \|t\|$  (voir figure 3.9).

Intuitivement, notre idée est d'utiliser  $\exp_p$  comme un projecteur tel que si on lui donne un vecteur tangent  $t = \vec{pq}$ , il nous fournit son projeté optimal  $\exp_p(t)$  de  $q$  sur la surface. Concrètement  $\exp_p$  s'obtient usuellement par une reparamétrisation de  $\gamma$  par la longueur d'arc. Pour cela, il faut calculer la longueur de la courbe  $s(t) = \int_0^t \|\frac{d\gamma}{dt}[x]\| dx$  avec  $\frac{d\gamma}{dt}(0) = t$ , puis exprimer  $t$  en fonction de  $s$ , et donc  $\gamma$  en fonction de  $s$ . Ainsi la géodésique correspondante est donnée par :  $\exp_p(t) = \gamma(\|t\|_p)$ . Le souci est que nous ne disposons pas d'une paramétrisation de  $\frac{d\gamma}{dt}$  en fonction de  $t$ , ce qui ne nous permet pas d'avoir une expression explicite de  $\gamma$  en fonction de  $s$  et donc de  $\exp_p$ .

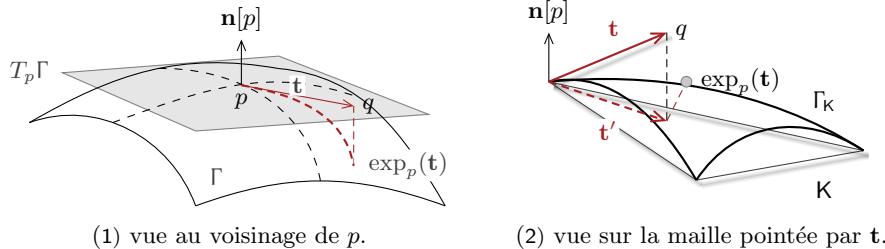


Figure 3.9: Approximation discrète de l'application exponentielle au voisinage d'un point.

ALGORITHME. La première étape consiste à déterminer quelle maille  $K$  incidente à  $p$  est pointée par  $t$ . En effet  $\exp_p(t)$  sera sur le patch quartique  $\Gamma_K$  associé à  $K$ . Ensuite la difficulté réside dans le choix d'un point  $\tilde{q} \in K$  telle que la longueur de la courbe géodésique  $\gamma$  sous-tendue soit égale à  $\|t\|$ . En d'autres termes, nous devons trouver un vecteur  $t' = [p\tilde{q}]$  à partir de  $t$  tel que  $\exp_p(t') = \Gamma_K[\tilde{q}]$ .

Dans notre cas, ce problème est résolu par une recherche linéaire à une étape. Pour trouver une valeur initiale de  $\tilde{q}$ , l'idée est d'abord d'effectuer une rotation de  $t$  sur le plan tangent  $T_K\Gamma$  de  $K$  comme montré sur la figure 3.9. Ensuite nous calculons le projeté  $\Gamma_K[\tilde{q}]$  de  $\tilde{q}$  sur  $\Gamma_K$ , ainsi que sa normale associée. Par la suite, l'idée est d'approcher le segment géodésique  $\gamma$  par une courbe B-spline sous-tendue par  $[p, \sigma_K[\tilde{q}]]$  et d'évaluer sa longueur. Cela va nous permettre d'ajuster la norme de  $t'$  par un facteur  $s$  de sorte que  $\|t'\| = \frac{\|t\|^2}{\ell[\gamma]}$ . Ainsi en re-calculant  $\Gamma_K[\tilde{q}]$  et  $\gamma$  on a bien  $\ell(\gamma) \approx \|t\|$ . La routine complète est décrite à l'algorithme 3.1.

<sup>34</sup>Sa réciproque est l'application logarithmique  $\log_p : \Gamma \rightarrow T_p\Gamma$ . Elle fournit le vecteur tangent associé à tout point d'une géodésique de la variété.

Algorithme 3.1: Approximation de  $\exp_p : T_p\Gamma \rightarrow \Gamma$

```

fonction EXPONENTIAL-MAP( $p, t$ )                                 $\triangleright t$  : le vecteur tangent à  $p$ .
    trouver  $K$  : la maille incidente à  $p$  pointée par  $t$ .
    si  $t$  est sur une arête  $c$  de  $K$  alors
        construire une B-spline  $\gamma$  sous-tendue par cette arête et retourner  $\gamma(\frac{\|t\|}{\|c\|})$ .
    sinon si  $t$  est interne à  $K$  alors
        construire le patch  $\Gamma_K$  et retourner le projeté  $\Gamma_K(p + t)$ .
    sinon
        calculer  $\theta$  l'angle entre  $t$  et  $t' \in T_K\Gamma$ .
        calculer la matrice de rotation  $R(\theta)$  qui ramène  $t$  sur  $T_K\Gamma$ .
        calculer  $q = p + R(\theta)t$  et paramétriser sur  $K$ .
        construire le patch  $\Gamma_K$  et calculer  $r = \Gamma_K[q]$  et sa normale  $n[r]$ .
        construire une B-spline  $\gamma$  sous-tendue par  $[pr]$ , puis calculer  $\ell(\gamma)$ .
        déduire  $t' = \frac{\|t\|^2}{\ell(\gamma)}$ , puis retourner le projeté  $\Gamma_K(p + t')$ .
    fin si
fin

```

### 3.2.4 Recherche locale

Pour résumer, nous avons identifié les entités topologiques à préserver, et nous disposons d'une orientation consistante de la surface ainsi qu'un opérateur de projection. Nous pouvons à présent définir les noyaux de recherche locale pour l'adaptation du maillage (figure 3.2, page 55). Dans notre cas, le **raffinement** et la **simplification** visent à contrôler le nombre de points de la triangulation selon d'une densité définie sur ses points. Par contre, la **bascule** et le **lissage** visent à optimiser les degrés des points ainsi que la forme et alignment des mailles selon la mesure de qualité décrite à (3.1) (page 57). Notons que tout remailleur local se base sur une combinaison de ces quatre noyaux (et leurs variantes). In fine, ils se différencient sur la manière dont ils tiennent compte des divers contraintes (localité, pas de dégradation excessive de mailles etc.). En fait, c'est justement ce qu'on cherche à décrire ici.

#### 3.2.4.1 Raffinement

TRAVAUX CONNEXES. Elle vise à enrichir la triangulation par insertion de points de STEINER selon une densité définie en ses points. Pour cela,

- La manière la plus simple et la plus intuitive consiste en une **bisection d'arêtes**. Pour chaque arête longue, on insère un point en son milieu et on le projette ensuite sur la surface. Sans contrôle explicite, cela tend à générer des mailles très étirées.
- Une manière de contourner ce problème serait d'utiliser un noyau de DELAUNAY [123, 124]. À quelques précautions près, il améliore strictement la qualité des mailles dans le voisinage du point en cours d'insertion. Le problème est que l'on ne peut pas borner la taille de la cavité du point à insérer. Ainsi on ne sait pas a priori le voisinage impacté par l'opération<sup>35</sup>, en particulier dans le cas anisotrope [123]. Même si elle est identifiée dans une boucle à part, la taille non négligeable de cette cavité réduit vraiment le nombre de tâches indépendantes à ordonner.

De ce fait, nous avons préféré utiliser le noyau de RIVARA [125]. Il a l'avantage d'être simple et local. En fait, on applique un motif de découpage différent selon le nombre d'arêtes longues de chaque maille (voir figure 3.2). Ici, on peut réduire la dégradation des mailles créées en choisissant convenablement comment on connecte les points de STEINER avec les sommets de la maille.

NOS CHOIX. À chaque insertion, le point est directement projeté sur la surface. Dans notre cas,

- Il aurait fallu que les sommets de la maille soient également re-projetés<sup>36</sup>, pour préserver la continuité  $G^1$ . Néanmoins cela implique de figer la boule de chaque sommet, ce qui réduit

<sup>35</sup>À vrai dire, ce noyau n'est pas trivialement parallélisable à grain fin, car c'est un algorithme irrégulier [168].

<sup>36</sup>C'est le cas des schémas classiques de subdivision de mailles, comme celui de Loop [112] par exemple.

considérablement la localité du noyau<sup>37</sup>. Ainsi nous avons préféré laisser ces sommets intacts durant cette phase, ils seront re-projetés uniquement au moment du lissage.

- Le sous-ensemble de mailles à raffiner ainsi que les motifs associés sont identifiés au préalable dans une boucle à part, ce qui induit un léger surcoût mémoire. Néanmoins, cela permet de gérer le stockage et l'indexation des points et mailles créées, afin de préserver le placement mémoire initial et limiter l'entrelacement des accès-mémoires en contexte multithread.
- L'approximation de la surface est toujours améliorée, car les patchs décrits à la section 3.2.3 représentent une région invariante de cette surface. Ainsi aucun contrôle d'erreur de la surface n'est nécessaire.

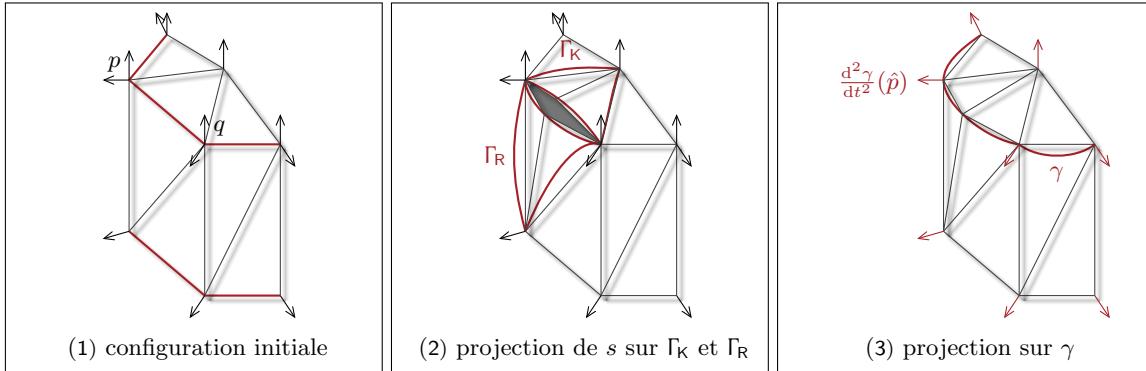


Figure 3.10: Raffinement d'une arête vive. En prenant les patchs  $\Gamma_K$  et  $\Gamma_R$  incidents à  $[pq]$ , on obtient deux projets différents du point de STEINER, ce qui induit un trou sur la surface (2). En considérant la B-spline relative à la courbe unique  $\gamma$  passant par  $[pq]$  on obtient un unique point de STEINER consistante avec la courbure de  $\gamma$  (3).

$\triangle$  RIDGES. Si l'un des points de STEINER est sur une arête vive  $[pq]$ , alors il doit être projeté de manière à ce que ses coordonnées soient unique, qu'il soit calculé à partir d'une maille K ou de l'autre maille R incidentes à cette arête. Ici, le point est projeté sur l'unique courbe de discontinuité  $\gamma$  sous-tendue par cette arête, par le biais de  $\exp_p(\frac{1}{2} \vec{p}\vec{q})$ . Dans notre cas, nous avons mémorisé les normales de  $\gamma$ <sup>38</sup>. Ainsi elle peut être construite de façon non ambiguë en utilisant les normales  $\mathbf{n}_i = \frac{d^2\gamma}{dt^2}(0)$  et  $\mathbf{n}_{i+1} = \frac{d^2\gamma}{dt^2}(1)$  (voir figure 3.10). D'un point de vue localité, cela permet de calculer et d'indexer chaque point de STEINER en amont, en n'utilisant que les données relatives aux coordonnées et normales des sommets de l'arête vive. Cela permet de traiter les deux mailles de part et d'autre de cette arête de manière indépendante comme montré sur la figure 3.10.

### 3.2.4.2 Simplification

TRAVAUX CONNEXES. Elle vise à réduire le nombre de points de la triangulation par suppression de points. En notant  $p$  le point à supprimer, et  $C$  la cavité résultant de la suppression de  $p$ , elle est usuellement réalisée par l'un des deux noyaux suivants :

- *décimation* [148] : ici la cavité est retriangulée de manière à minimiser la dégradation des mailles et à équilibrer les degrés des points. Le problème est que le nombre de configurations possibles croît exponentiellement<sup>39</sup> en la taille de la cavité, et il faut toutes les tester avant de trouver l'optimale. Ainsi l'inconvénient majeur est qu'on ne peut pas prédire quelles données d'incidence doivent être mises à jour parmi les points de la cavité<sup>40</sup>.

<sup>37</sup>En effet le nombre de mailles impactées serait de  $k = \sum_{i=1}^3 [\delta(p_i) - 1]$ .

<sup>38</sup>En fait, nous avons mémorisé leurs dérivées secondes  $\frac{d^2\gamma}{dt^2}(p_i)$  dans une table  $\mathbf{r}[\mathbf{M}[i]]$ , où  $\mathbf{M}$  est le mapping surjectif qui associe l'indice d'un ridge ou coin de  $(P, M, n)$  au graphe d'adjacence des ridges et coins  $(R, S, \beta)$  (voir page 58).

<sup>39</sup>notamment dans le cas anisotrope.

<sup>40</sup>Du coup, ils doivent donc tous être figés en contexte multithread.

- *contraction* [149] : cette fois, le point est fusionné avec un voisin  $q$ , et le point résultant est ensuite repositionné sur la variété. L'avantage est qu'on a un contrôle implicite de l'erreur d'une part, et qu'on sait exactement quelles données doivent être mises à jour après la suppression du point d'autre part. En effet, on choisit de repositionner le point de manière à minimiser la dégradation induite par la suppression de  $p$ <sup>41</sup>. Bien que simple et efficace<sup>42</sup> comparé au noyau précédent, il n'est néanmoins pas exempt d'inconvénients. En effet le degré du point résultant croît strictement, ce qui accroît l'irrégularité de la triangulation<sup>43</sup> (figure 3.11). Enfin, ce noyau requiert quelques précautions car il peut induire des inconformités topologiques.

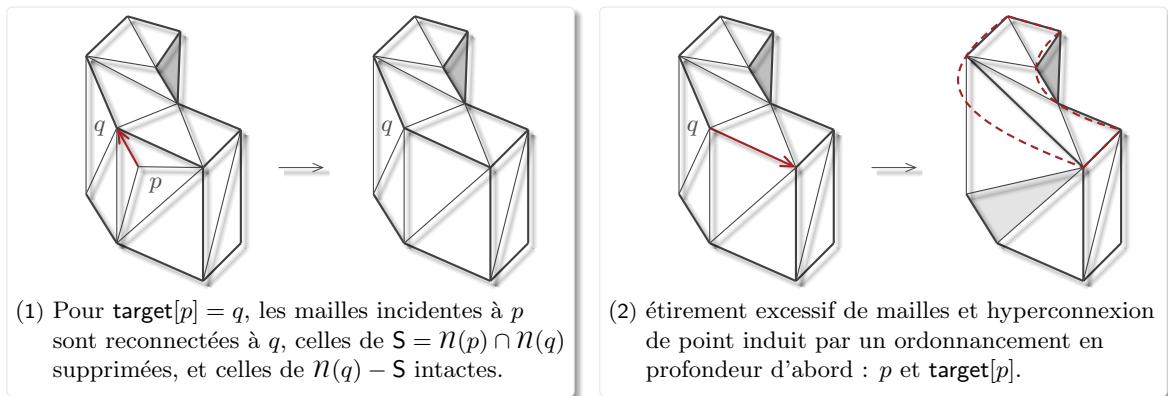


Figure 3.11: Simplification par fusion de points.

NOS CHOIX. Dans notre cas, nous avons opté pour la fusion de points puisqu'elle n'implique que le voisinage du point à supprimer en séquentiel, et que les données d'incidence à mettre à jour sont statiques et minimales. Comme un certain nombre de vérifications doivent être effectuées avant la validation des modifications, nous décidons de scinder le traitement en deux passes<sup>44</sup> :

- **filtrage.** Ici l'idée est de trouver une correspondance injective  $\text{target}$  entre les points à supprimer et leurs voisins cibles, conformément à la densité prescrite sur la triangulation. Pour chaque candidat  $p$ , on examine son voisinage afin de trouver le voisin optimal  $q$  tel que :

- l'intersection des voisinages de  $p$  et  $q$  ne contient que les points opposés à  $[pq]$ ,
- la dégradation de la surface est minimale avec  $q$  comparée à tout autre voisin,
- la déviation des normales aux mailles incidentes à  $q$  par rapport à  $\mathbf{n}[q]$  n'excède pas le seuil angulaire  $\theta_{\max}$  spécifié<sup>45</sup>.

Notons qu'il se peut qu'aucun point cible n'ait été trouvé pour un point  $p$  donné, et donc que  $\text{target}[p] = \emptyset$ . Sinon tous les points candidats  $p$  tel que  $\text{target}[p] \neq \emptyset$  sont regroupés et triés par ordre croissant de leurs erreurs locales accumulées sur tous les itérés. Cela permet de contrôler le nombre de points de la triangulation conformément au budget de points fixé.

- **application.** La mise à jour proprement dite de la triangulation consiste à reconnecter les mailles incidentes à  $p$  vers  $q$ , puis à repositionner  $q$ . Pour cela, les mailles de  $S = N[p] \cap N[q]$  sont supprimées, et toute référence à  $p$  dans  $N[p]$  est remplacée par  $q$ . Le point résultant est ensuite projeté sur la surface avec  $\exp_p(\frac{1}{2} \vec{pq})$ . Dans notre cas, la mise à jour des données d'incidence :

- est minimale<sup>46</sup> : seuls  $q$  et les deux points opposés à  $[pq]$  sont concernés.

<sup>41</sup>En fait, c'est le plus utilisé en simplification de surfaces à l'instar du noyau basé sur l'erreur quadrique [115].

<sup>42</sup>en termes de nombres d'opérations.

<sup>43</sup>De plus, les voisinages à la fois de  $p$  et de  $q$  doivent être figés en contexte multithread (au lieu d'un seul)

<sup>44</sup>Notons que la suppression de deux points non voisins vers un même point peut être problématique en multithread : sans arbitrage, on ne sait pas où projeter le point résultant dans ce cas. Nous verrons comment le gérer plus tard.

<sup>45</sup>Cela permet également de vérifier qu'on n'a pas un repliement de la surface.

<sup>46</sup>Et cela qu'importe la taille des voisinages de  $p$  ou  $q$ . Par ailleurs, on ne peut pas faire moins que cela.

■ est statique<sup>47</sup> : elle consiste à supprimer  $S$  des voisinages de ces trois points.

Le noyau est ensuite propagé en marquant chaque voisin de  $q$  comme étant candidat.

Enfin, notons que cette structuration en deux passes force un ordonnancement du graphe de tâches en largeur d'abord. Cela permet d'éviter l'étirement excessif des mailles et un degré excessif de points due à une contraction récursive (ou en profondeur), comme montré à la figure 3.112.

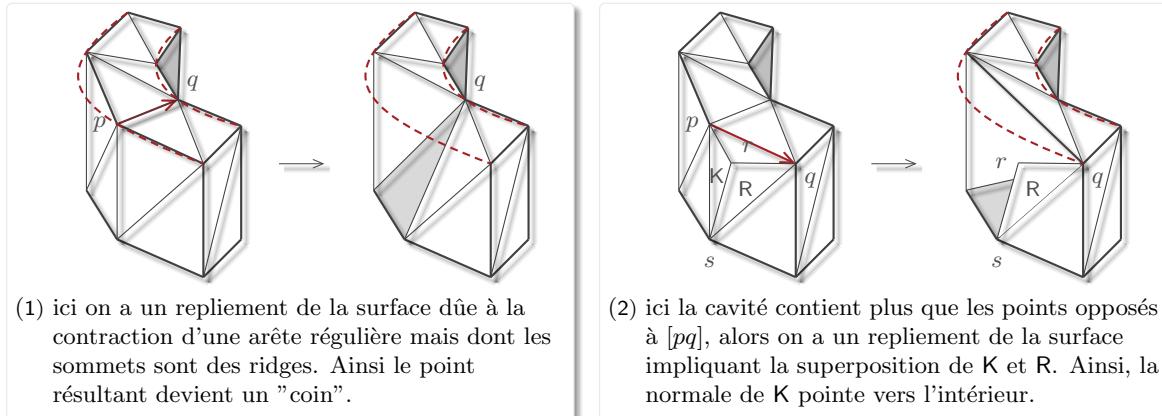


Figure 3.12: Simplifications interdites.

**RIDGES.** Les "coins" ne doivent pas être traités, tandis que si le point à supprimer  $p$  et son voisin  $q$  sont des "ridges", alors  $[pq]$  ne peut être contractée que si c'est une arête vive. Sinon elle peut induire un repliement de la surface triangulée qui ne serait plus une variété discrète (figure 3.12). D'un point de vue implémentation, cette contrainte est assez forte dans la mesure où elle implique d'extraire et de maintenir l'adjacence des ridges et coins, ce qui induit un certain nombre d'indirections. De plus, elle est fastidieuse puisque les ridges sont dupliqués dans notre cas. Au moment du filtrage,

- les deux points cibles d'un candidat  $p_i$  sont extraits à partir du graphe  $(R, S, \beta)$  avec  $(q_{1,k}, q_{2,k})_{k=1}^2 = (R[i^-], R[i^+])$  où  $\{i^-, i^+\} = \beta(M[i]) \subsetneq R \cup S$  (voir page 58).
- les paires de points originaux  $q_{j,1}$  et dupliqués  $q_{j,2}$  sont également extraites pour chaque région  $\Gamma_1$  et  $\Gamma_2$  délimitée par l'arête vive.
- la contraction est simulée sur les quatre sous-voisinages  $n[q_{j,k}]_{j,k=1}^2$ . En notant  $p_z$  le dupliqué de  $p_i$  dans le graphe  $(P, M, n)$ , le point cible  $q_j$  n'est retenu que si la contraction est valide dans les deux sous-voisinages relatifs à  $\Gamma_1$  et  $\Gamma_2$ , c'est-à-dire que  $\text{target}\{p_i, p_z\} = \{q_{j,1}, q_{j,2}\}$ .

Enfin, la mise à jour de  $(P, M, n)$  est effectuée comme dans le cas d'un point régulier mais sur les deux sous-voisinages du point cible original et dupliqué. Le graphe  $(R, S, \beta)$  est également mis à jour en connectant le point cible avec le point non-retenu mais dans l'espace d'indexation  $R \cup S$ . La procédure complète est résumée à l'algorithme 3.3 en notant  $\nu = M$  et  $\nu^{-1} = R$  pour plus de clarté.

### 3.2.4.3 Relaxation

**TRAVAUX CONNEXES.** Selon le contexte, elle vise à régulariser le degré des points, ou bien à améliorer la qualité des mailles par reconnection des points de la triangulation. Elle est souvent réalisée par des basculement d'arêtes, mais peut impliquer d'autres noyaux. Dans notre cas, on s'intéresse à la relaxation des degrés des points car les interpolations numériques sont plus stables et précises sur un maillage quasi-structuré<sup>48</sup>. En notant  $d$  le degré d'un point, et  $d^*$  son degré optimal<sup>49</sup>, il s'agit de résoudre :

$$\min_{\mathcal{T}_h} \mathcal{R}(\mathcal{T}_h) = \min_{\mathcal{T}_h} \|d - d^*\|_2 = \min_{(P, M, n)} \left[ \sum_{i=1}^n (d[p_i] - d^*)^2 \right]^{\frac{1}{2}} \quad (3.12)$$

<sup>47</sup>Ce point est crucial pour l'implémentation d'un mécanisme de synchronisation en contexte multithread.

<sup>48</sup>Dans notre cas, l'amélioration de la qualité est reportée à l'étape de lissage.

<sup>49</sup>Le degré optimal d'un point est 6 s'il est interne à la surface et 4 s'il est sur le bord d'une surface.

Bien que d'une formulation simple, le problème d'optimisation combinatoire en (3.12) est loin d'être trivial, puisqu'on tombe souvent sur des minima locaux et que la distance entre deux solutions est souvent importante. De plus on trouve peu de travaux relatifs à ce problème dans la littérature, et aucune des heuristiques existantes ne garantit de fournir l'optimum global, ou à défaut un bon ratio d'approximation. Parmi les travaux récents, on peut distinguer :

- le schéma 5-6-7 [135, 136]. Il garantit que le degré de tout point régulier soit compris entre 5 et 7, quelque soit le *genre* de la surface. Ici, on a un motif à appliquer selon le degré du point (en se basant sur la caractéristique d'EULER de la surface [12], §1.1), et nécessite un rééchantillonnage local. L'inconvénient majeur réside dans le surcoût mémoire induit par un nombre important de mailles générées. De plus il entrelace des noyaux dédiés à des buts différents sur des voisinages non restreints, les rendant difficile à contrôler en contexte multithread.
- le *puzzle solving* [120]. À partir d'un minimum local, chaque arête irrégulière est classée selon le degré de ses sommets :  $\{\mathbf{e}_i^+, \mathbf{e}_i^-, \mathbf{e}_i^0\}$ . Ici les  $\mathbf{e}_i^+$  et  $\mathbf{e}_i^-$  sont respectivement l'ensemble des arêtes dont les deux sommets ont un degré supérieur (respectivement inférieur) à 6; et  $\mathbf{e}_i^0$  l'ensemble d'arêtes dont le degré d'un des deux sommets est supérieur à 6, et l'autre inférieur à 6. Le puzzle est ensuite résolu en raffinant les  $\mathbf{e}_i^+$ , en contractant les  $\mathbf{e}_i^-$ , et en déplaçant les  $\mathbf{e}_i^0$  par bascule d'arêtes jusqu'à ce qu'elles deviennent des  $\mathbf{e}_i^+$  ou des  $\mathbf{e}_i^-$ . Notons que les nombres d'arêtes à traiter et de mailles créées sont moindres comparés au schéma 5-6-7. Bien qu'élégante, cette approche est néanmoins difficile à paralléliser car on ne peut pas borner la propagation des  $\{\mathbf{e}_i^0\}$ .

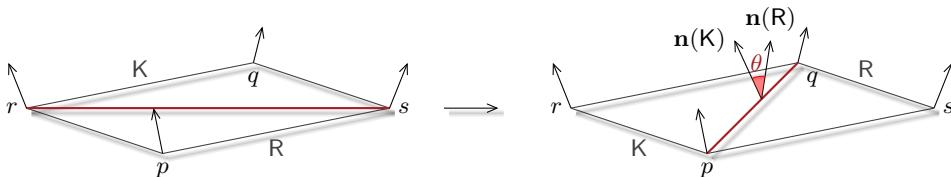


Figure 3.13: Relaxation de degrés par bascule d'arête. Notons que le polygone  $(p, q, r, s)$  est convexe. La non-dégénérescence des mailles résultantes ainsi que la déviation de leurs normales est vérifiée après application du noyau.

NOS CHOIX. In fine, on est amené à trouver un graphe biparti  $(P, M, \mathcal{N})$  qui minimise  $\mathcal{R}$ . En fait nous ne souhaitons ni ré-échantillonner la surface, ni entrelacer les noyaux dédiés à des buts différents entre eux. En effet cela compliquerait leur contrôle (propagation) au moment de les paralléliser. Du coup, nous avons décidé de ne contrôler que la variation du degré de chaque point en modifiant les arêtes qui leur sont incidentes (figure 3.13). À l'itéré  $t$ , on bascule l'arête commune à deux mailles :

- si le degré des quatre sommets des deux mailles est en moyenne réduit,
- si la qualité de la pire maille est améliorée,
- si la déviation des mailles aux plans tangents de chaque point n'excède pas un seuil  $\theta_{\max}$ .

Une manière naturelle de l'implémenter est de parcourir tous les points et de traiter les arêtes incidentes. Dans notre quête de localité, on décide de traiter plutôt par paires de mailles, ce qui minimise les mises à jour des données d'incidence. Notons enfin que ce noyau ne nécessite pas de traitement particulier des arêtes vives : elles sont juste ignorées. La procédure complète est résumée à l'algorithme 3.4, et son impact sur l'irrégularité de la triangulation est illustré à la figure 3.23.

### 3.2.4.4 Lissage

TRAVAUX CONNEXES. Il vise à débruiter la surface à l'issue du rééchantillonnage, mais aussi à améliorer la qualité des mailles sans changer la connectivité des points. Il existe une vaste littérature sur le lissage, et un aperçu est donné dans [1](§4) dans le cas d'un pipeline de traitement graphique. Néanmoins les approches peuvent être regroupées en deux classes

- *diffusion*. Elle regroupe le lissage laplacien et toutes ses variantes. Ici la surface est vue comme un signal  $\mathbf{f}$ , et le lissage consiste à débruiter  $\mathbf{f}$  par application d'un filtre de diffusion basé sur

son laplacien  $\Delta f$  [150]. Ainsi la position de chaque point  $p$  est mise à jour en fonction de celles de ses  $n$  voisins :

$$p \leftarrow p + \lambda \Delta^k p, \text{ avec } \begin{cases} \lambda \in \mathbb{R}^+ : \text{facteur d'échelle}, \\ \Delta^1 p = \sum_{j=1}^n \omega_j (p_j - p), \omega_j \in [0, 1] \\ \Delta^k p = \Delta(\Delta^{k-1} p), \forall k \geq 2 \end{cases} \quad (3.13)$$

où les coefficients normalisés  $\omega_j$  dépendent de la méthode de discréétisation du laplacien. En fait c'est le choix des  $k$ ,  $\lambda$  et surtout des  $\omega_{i,j}$  qui va discriminer les variantes<sup>50</sup>. In fine, le point est repositionné au barycentre pondéré de son voisinage. Ce noyau a l'avantage d'être simple, et il permet à la fois de régulariser la forme de la surface, ainsi que celle des mailles [154]. Par contre,

- il tend à rétrécir la variété car ce n'est pas un filtre passe-bas
- il n'améliore pas strictement la qualité des mailles car c'est une heuristique.

Ce retrécissement peut néanmoins être compensé en alternant  $\lambda$  par un facteur négatif de gonglement  $\lambda^-$  au cours des itérations par exemple [116].

- *optimisation.* Ici le déplacement d'un point n'est plus calculé de manière heuristique mais dicté par la minimisation d'une fonction cout  $f$ . Pour chaque point  $p$ , il consiste à résoudre le problème d'optimisation sous contraintes suivant :

$$p \leftarrow \min_{p \in C} f(p) \text{ avec } \begin{cases} C = \bigcup_{j=1}^n \Gamma_j : \text{voisinage continu de } p, \\ \Gamma_j : \text{patch de la maille } K_j \text{ incidente à } p. \end{cases} \quad (3.14)$$

Un tel noyau fournit des features avancées comme :

- la préservation du volume intérieur lors d'un débruitage par exemple [155, 156].
- l'amélioration de la pire maille dans les cas où un laplacien échoue<sup>51</sup> [157, 158, 113].

Pour cela, une direction de descente est calculée, et le point est déplacé graduellement jusqu'à atteindre un minimum local. L'inconvénient est que cela engendre des calculs coûteux<sup>52</sup>.

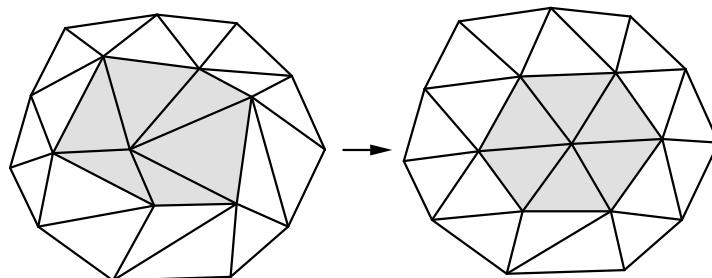


Figure 3.14: But du lissage dans notre contexte.

NOS CHOIX. Dans notre cas, le lissage vise à améliorer la qualité des mailles comme sur la figure 3.14. En fait c'est le seul noyau dédié à cette tâche parmi les quatre. Il doit donc être précis sans pour autant être trop coûteux. À cette fin, nous avons opté pour un noyau mixte comme dans [158, 159], afin de tirer avantage des deux familles d'approches. L'idée est d'appliquer un noyau de diffusion en premier lieu, puis de recourir à un noyau d'optimisation en cas d'échec. Pour ce dernier, nous considérons la mesure de forme décrite à la page 57 comme l'inverse de notre fonction objectif.

- **DIFFUSION.** Pour chaque point, le but est d'égaliser l'aire des patchs<sup>53</sup> des mailles incidentes, tout en minimisant la déformation de la surface. Pour cela, l'idée est de déplacer chaque point

<sup>50</sup>Notamment le noyau bi-laplaciens [151], basé sur la courbure moyenne [152], ou sur une diffusion anisotrope [153].

<sup>51</sup>Comme le cas des voisinages concaves par exemple.

<sup>52</sup>Selon la fonction objectif, il peut induire des techniques non triviales (et donc coûteuses) ou lentes à converger.

<sup>53</sup>Le calcul de l'aire d'un patch est donné en annexe, page 134.

vers le projeté du centre de masse pondéré de son voisinage sur la surface, en tenant compte de la densité prescrite en ces points, voir (3.15). On calcule d'abord la direction de déplacement  $\mathbf{t}$  vers le centre de masse de la région  $C = \bigcup_{j=1}^n \Gamma_j$ . Ensuite on calcule le segment courbe géodésique  $\gamma$  d'origine  $p_i$  et de vitesse initiale  $\mathbf{t}$ . Enfin, la nouvelle position de  $p$  est donnée par  $\gamma(1)$ .

$$p = \exp_p \left[ \alpha \frac{\int_C \log_{p^{[t]}}[x] \rho[x] dx}{\int_C \rho[x] dx} \right], \text{ avec } \begin{cases} C : \text{voisinage continu de } p, \\ \rho : \text{fonction de densité,} \\ \alpha : \text{facteur d'échelle.} \end{cases} \quad (3.15)$$

Ici,  $\rho[x] = \sqrt{\det[J_x^\top g_x J_x]}$  vise à pondérer le déplacement de  $p$  conformément au tenseur métrique  $g_x$  de chaque point au voisinage de  $x$ , et  $J_x$  la matrice jacobienne de la paramétrisation de la surface au point  $x$  (voir page 133). Afin d'éviter des calculs inutiles, on exclut d'embellie le cas où le centre de masse est trop éloigné du point à déplacer<sup>54</sup>. Enfin, avant de valider le déplacement du point, on s'assure que :

- la qualité de la pire maille incidente à  $p$  soit strictement améliorée.
- la déviation des mailles incidentes à  $p$  n'excède pas le seuil angulaire  $\theta_{\max}$ .

Notons qu'ici l'opération s'effectue directement sur la surface et pas dans le plan tangent  $T_p\Gamma$  de  $p$  contrairement à [132] par exemple. En effet la paramétrisation obtenue par projection des mailles incidentes à  $p$  sur  $T_p\Gamma$  peut engendrer de fortes distorsions. Ainsi la position calculée sur  $T_p\Gamma$  ne correspond pas forcément au centre de masse une fois projetée vers la surface.

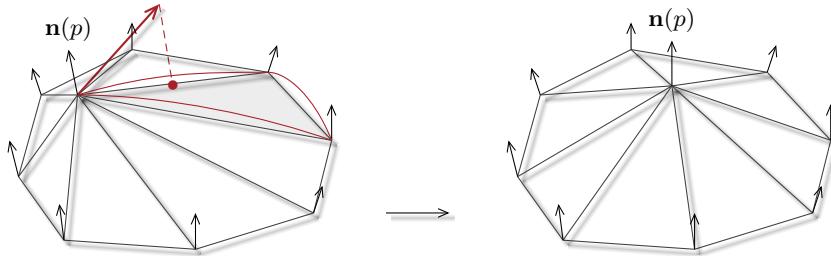


Figure 3.15: Lissage par le noyau de diffusion.

- **OPTIMISATION.** Le but est de forcer l'amélioration de la qualité de la pire maille incidente à  $p$  : la fonction coût en (3.14) correspond à la distorsion de cette maille. Notons  $f_j[p]$  la distorsion<sup>55</sup> d'une maille  $K_j$  incidente à  $p$  selon la position actuelle de  $p$ . Ainsi le but est de minimiser  $f_k[p] = \max_j f_j[p]$  avec la contrainte que  $p$  doit rester sur  $C$ . Puisqu'on dispose d'une méthode de projection explicite (voir page 64), nous décidons d'utiliser une méthode de gradient projeté<sup>56</sup>. Pour cela, l'idée est de déplacer  $p$  graduellement sur la surface vers la direction opposée à son gradient, selon un pas de déplacement variable  $\alpha$ , et cela jusqu'à convergence ou si un seuil sur le nombre d'itérés n'est atteint. À l'itéré  $t + 1$ , la position du point  $p$  est mise à jour par :

$$p^{[t+1]} = \exp_{p^{[t]}}[-\alpha \nabla f_k(p^{[t]})], \text{ avec } \begin{cases} f_j : \text{dégradation de la maille } K_j \\ f_k = \max_j f_j[p^{[t]}] \\ \alpha : \text{pas de déplacement} \end{cases} \quad (3.16)$$

Pas initial. Ici, minimiser  $f_k[p]$  peut accroître la distorsion des autres mailles. Ainsi, nous devons en tenir compte lors du choix du pas initial à chaque itéré. Pour cela, nous observons la variation de  $f_k$  et  $f_{j \neq k}$  quand la position de  $p$  est mise à jour. Plus précisément, considérons le développement de Taylor-Young de  $f_k$  et  $f_j$  à l'ordre un, au voisinage de  $p$ , qui s'écrit :

$$f_j(p_i - \alpha \nabla f_k(p_i)) = f_j(p_i) + \langle -\alpha \nabla f_k(p_i), \nabla f_j(p_i) \rangle + o(-\alpha \|\nabla f_k(p_i)\|) \quad (3.17)$$

<sup>54</sup>Et précisément si le centre de masse n'est pas inclus dans la sphère circonscrite à  $K$ . Cela correspond à un cas pathologique qui sera traité par la routine suivante.

<sup>55</sup>La distorsion d'une maille est juste l'inverse de sa qualité.

<sup>56</sup>Elle a l'avantage d'être simple avec les mêmes garanties de convergence que dans le cas sans contrainte.

Ainsi si on veut préserver la qualité des autres mailles, on doit donc choisir le pas de manière à minimiser à la fois  $f_k$  et  $f_j$ , et notamment l'écart entre  $f_k(p)$  et  $f_j(p)$ . Cela s'obtient en résolvant l'équation suivante :

$$f_\nu(p_i - \alpha \nabla f_k(p_i)) = f_k(p_i - \alpha \nabla f_k(p_i)), \text{ avec } \nu = \operatorname{argmin}_{j \neq k} f_j(p_i) \quad (3.18a)$$

$$f_\nu(p_i) - f_k(p_i) = \langle -\alpha \nabla f_k(p_i), \nabla f_\nu(p_i) \rangle - \langle -\alpha \nabla f_k(p_i), \nabla f_k(p_i) \rangle \quad (3.18b)$$

$$f_\nu(p_i) - f_k(p_i) = -\alpha \langle \nabla f_k(p_i), \nabla f_\nu(p_i) \rangle + \alpha \|\nabla f_\nu(p_i)\| \quad (3.18c)$$

$$\boxed{\alpha = \frac{f_\nu(p_i) - f_k(p_i)}{\|\nabla f_k(p_i)\| - \langle \nabla f_k(p_i), \nabla f_\nu(p_i) \rangle}} \quad (3.18d)$$

**Prochain pas.** Ensuite, le prochain pas est déterminé itérativement par une recherche linéaire vérifiant les critères de WOLFE. Ils garantissent que  $f_k$  décroît significativement (3.19), et que  $\alpha$  est suffisamment grand pour converger rapidement (3.20). Ils s'écrivent :

$$f_k(p_i - \alpha \nabla f_k(p_i)) \leq f_k(p_i) - \omega_1 \alpha \|\nabla f_k(p_i)\| \quad (3.19)$$

$$\langle \nabla f_k(p_i - \alpha \nabla f_k(p_i)), -\nabla f_k(p_i) \rangle \geq \omega_2 \|\nabla f_k(p_i)\| \quad (3.20)$$

En posant  $(\alpha_{\min}, \alpha_{\max}) = (0, \infty)$  ainsi que deux paramètres  $0 < \omega_1 < \omega_2 < 1$ , le prochain pas  $\alpha$  est déterminé à l'issue des tests suivants :

- si (3.19) n'est pas vérifiée, alors on diminue  $\alpha_{\max} = \alpha$  et  $\alpha = (\alpha_{\min} + \alpha_{\max})/2$ .
- si (3.20) n'est pas vérifiée, alors on augmente  $\alpha_{\min} = \alpha$  et  $\alpha = \begin{cases} 2\alpha_{\min} & \text{si } \alpha_{\max} = \infty \\ (\alpha_{\min} + \alpha_{\max})/2 & \text{sinon} \end{cases}$
- sinon c'est ok.

La routine complète est illustrée et résumée à la figure 3.16. Avant de valider le déplacement, nous vérifions que la déviation des mailles au plan tangent de  $p$  n'excède pas le seuil  $\theta_{\max}$  fixé<sup>57</sup>.

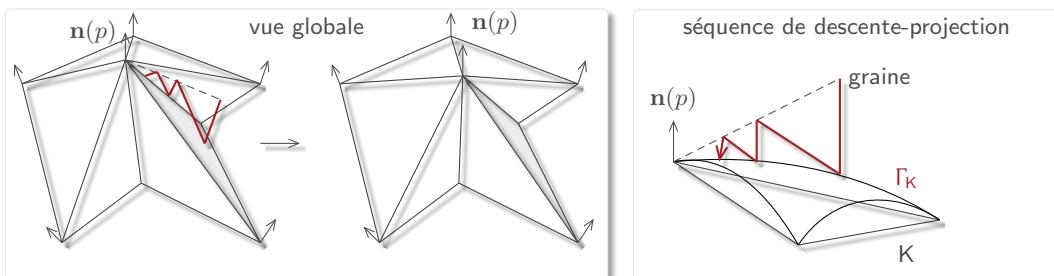


Figure 3.16: Lissage par optimisation non-linéaire. Partant d'un point initial, on se déplace pas à pas sur la surface tant que la distorsion  $f_k(p)$  de la pire maille incidente à  $p$  décroît. À chaque itéré, la direction est donnée par  $\mathbf{t} = -\nabla f_k(p)$  et le pas optimal  $\alpha$  est calculé par une recherche linéaire à partir de la distorsion  $f_j(p)$  des autres mailles. Le point est projeté sur le patch relatif à la maille pointée par  $\alpha \mathbf{t}$ .

**RIDGES.** Si  $p$  est un ridge alors il est déplacé sur l'unique courbe de discontinuité  $\gamma$  passant par  $p$ . En fait, le principe reste le même que pour un point régulier, mais les calculs sont simplifiés puisqu'ils se font sur une courbe et non plus à une surface. Ici le point est déplacé sur  $\gamma$  comme suit :

$$p = \alpha \frac{\int_\gamma x \rho[x] dx}{\int_\gamma \rho[x] dx}, \text{ avec } \begin{cases} \gamma : \text{courbe passant par } [pq] \text{ et } [rp], \\ \rho : \text{fonction de densité,} \\ \alpha : \text{facteur d'échelle.} \end{cases} \quad (3.21)$$

Ici,  $[pq]$  et  $[rp]$  sont les deux arêtes vives incidentes à  $p$ , et  $\gamma$  est approchée par une B-spline quartique. Enfin la routine de descente relative à (3.14) reste identique sauf que la projection est contrainte sur  $C = \gamma$  cette fois<sup>58</sup>.

<sup>57</sup>Notons que cela aurait pu être intégré aux contraintes, mais cela impliquerait de changer carrément d'algorithme.

<sup>58</sup>Néanmoins, on se restreint au noyau de diffusion pour les ridges en pratique.

Algorithme 3.2: Raffinement

**assure que :**

- chaque nouveau point est correctement projeté sur la surface idéale.
- chaque ridge dispose d'une normale et d'un tenseur métrique par région.

**1. Filtrage des mailles, référencement des points de Steiner.**

**pour** chaque maille  $K_i$  **faire**

    pattern[i] = 0.

**pour** chaque arête [pq] de  $K_i$  **faire**

**si**  $\ell_h(pq) > \ell_{\max}$  **alors**

            incrémenter pattern[i] ainsi que n.

            ▷ n :  $|\mathcal{T}_h|$

            calculer explicitement le point  $s = \exp_p(\frac{1}{2}\vec{pq})$ .

            ▷ indexation de s.

**si** c est une arête-vive et  $M[p] \leq M[q]$  **alors**

            résoudre les indices uid, orig et twin de s et incrémenter m.

            ▷ m :  $|R \cup S|$

            mettre à jour  $M[\text{orig}] = M[\text{twin}] = m$ .

            mettre à jour  $R[\text{uid}] = \{\text{orig}, \text{twin}\}$ .

**si** p ou q est un "coin" **alors**

            stocker steiner[hash(p, q)] = orig.

**sinon**

        ▷ duplication d'indices

$([i_p, j_p], [i_q, j_q]) = (R[M[p]], R[M[q]])$ .

        stocker steiner[hash[min(i<sub>p</sub>, i<sub>q</sub>), max(i<sub>p</sub>, i<sub>q</sub>)] = orig.

        stocker steiner[hash[min(j<sub>p</sub>, j<sub>q</sub>), max(j<sub>p</sub>, j<sub>q</sub>)] = twin.

**fin si**

calculer la normale  $r[s] = -\gamma_N(\frac{1}{2})$ .

**sinon si** p ≤ q **alors**

stocker steiner[hash(p, q)] = n.

calculer la normale n[s] et la métrique g<sub>s</sub>.

**fin si**

**fin si**

**fin**

**fin**

**2. Application du pattern**

**pour** chaque maille  $K_i$  **faire**

mémoriser  $s_j = \text{steiner}[\text{hash}(p_{j+1}, p_{j+2})]$ ,  $\forall 0 \leq j \leq 2$ .

▷ indices modulo 3.

**pour** chaque  $s_k$  valide **faire**

**si** c<sub>k</sub> est une arête vive **alors**

calculer la normale n[s<sub>k</sub>] et la métrique g<sub>s<sub>k</sub></sub>.

**fin**

**si** pattern[i] = 1 et  $\exists j : s_j$  valide **alors**

créer et stocker les deux mailles.

▷ écraser  $K_i$  et maintenir un offset off.

mettre à jour  $n[p_j], n[p_{j+2}]$  et  $n(s_j)$ .

▷  $n[p_{j+1}]$  est gardé intact.

**sinon si** pattern[i] = 2 et  $\exists j : s_j$  invalide **alors**

créer les trois mailles en choisissant la diagonale la plus courte.

mettre à jour les  $n[p_k]$  et  $n(s_k)$  pour tout  $k \neq j$ .

**sinon**

créer et stocker les quatre mailles.

mettre à jour  $n[p_1], n[p_2]$  et  $n(s_k)$  pour tout k.

**fin si**

**fin**

© 2018. HOBY RAKOTOARIVELO

## Algorithme 3.3: Simplification

**assure que :**

- les trois conditions de liens et la contrainte de continuité  $G^1$  sont respectées,
- la dégradation de la surface est minimale compte-tenu de  $n_{\max}$ ,
- un ordonnancement en largeur d'abord est appliqué.

**répéter**

1. Filtrage des points cibles optimaux ▷  $S = N[p] \cap N[q]$ .  
**pour** chaque point  $p$  **faire** ▷  $C = N[p] \cup N[q]$ .

**si**  $p$  est régulier **alors**

score : table des erreurs locales de  $p$ .  
**pour** chaque voisin  $q$  tel que  $|S| = 2$  **faire**  
simuler la contraction de  $p$  avec  $q$ .  
si valide, rajouter score $[q] = \varepsilon_h(q)$ .  
**fin**  
stocker target $[p] = \arg \min_i \text{score}[i]$ . ▷ target $[p]$  peut être invalide.

**sinon si**  $p$  est un "ridge" **alors**

score $[i]$  : table 2D des erreurs locales de  $p$ .  
 $C$  : mapping de  $p$  avec ses cibles potentielles. ▷ table 2D.  
extraire  $[p_0, p_1] = R[M[p]]$ . ▷ on ne sait pas si  $p$  est l'original ou le dupliqué.  
extraire  $[i^-, j^-] = R[\beta[M[p]]^-]$ . ▷ indice original et dupliqué de prev $[p]$ .  
extraire  $[i^+, j^+] = R[\beta[M[p]]^+]$ . ▷ idem pour next $[p]$ .  
**pour** chaque voisin  $q$  tel que  $|S| = 2$  **faire** ▷  $C[-/+][i \text{ ou } j]$ .  
simuler la contraction de chaque  $p_i$  avec  $C[q][i]_{i \in 0,1}$ .  
**si** les deux sont valides **alors**  
| rajouter score $[i][C[q][i]] = \varepsilon_h(C[q][i])_{i \in 0,1}$ .  
**fin**  
stocker target $[p_i] = \arg \min_j \text{score}[i][j]_{i \in 0,1}$ .

**fin si**  
**fin**

2. Application des modificationstasks : file des paires  $(i, \text{target}[i])$  valides par ordre croissant de  $\varepsilon_h(\text{target}[i])$ .**pour** chaque paire  $[p, q] \in \text{tasks}$  **faire**

remplacer les occurrences de  $p$  par  $q$  pour chaque maille incidente à  $p$ .  
supprimer  $S$ , modifier  $N[q] = C - S$   
enlever  $S$  de  $N[r]$  pour tout point  $r$  opposé à  $[pq]$ .  
**si**  $[pq]$  était une arête vive **alors**  
| mettre à jour  $\beta[M[q]]$  et  $\beta[M[r]]$  où  $r = R[\beta[M[p]]]$ . ▷ la courbe  $\gamma$  passe par  $r, p, q$ .

**fin**

**jusqu'à** nombre de points :  $n_{\max}$ 

**fonction** SIMULER( $p, q$ ) ▷ simule la fusion de  $p$  vers  $q$ .

**pour** chaque maille  $K$  incidente à  $p$  **faire** ▷ simule la fusion de  $p$  vers  $q$ .

remplacer l'occurrence de  $p$  dans  $K$  par  $q$ ,  
projeter  $q = \exp_p(\frac{1}{2}\vec{pq})$  et calculer  $\mathbf{n}(K)$ .  
vérifier que  $\det[J_K] \geq \epsilon$  et que  $q[K] \geq q_{\min}$ . ▷ non dégénérescence.

vérifier que  $\langle \mathbf{n}[p_i], \mathbf{n}[K] \rangle \in [0, \cos(\theta_{\max})]$ ,  $\forall p_i \in K$ . ▷ inversion ou déviation excessive.

projeter  $K$  sur  $T_q\Gamma$ , et vérifier que  $\det[J_K] \geq \epsilon$ . ▷ distorsion excessive sur  $T_q\Gamma$ .

vérifier que  $\ell_h(\mathbf{c}_i) \leq \ell_{\max}$ , pour toute arête  $\mathbf{c}_i \in K$ . ▷ étirement excessif.

calculer la contribution de  $K$  à  $\varepsilon_h[q]$  et remettre  $p$  dans  $K$ .

**fin**  
**retourner**  $\varepsilon_h[q]$  si valide, et  $-1$  sinon.  
**fin**

Algorithme 3.4: Relaxation des degrés.

**assure que :**

- la moyenne des degrés des points tend vers  $d^*$ .
- la contrainte de continuité  $G^1$  est respectée.

marquer toutes les arêtes comme étant à traiter

**répéter**

```

pour chaque maille M : ( $p_0, p_1, p_2$ ) faire
    pour chaque arête c de M marquée faire
        si c est ni vive ni frontière alors
            extraire  $p_3$  le point opposé à c et calculer  $\chi_{[M]} = \sum_{i=0}^3 |\deg[p_i] - d_i^*|^2$ .
            simuler la bascule de c et former les mailles résultantes  $K_1$  et  $K_2$ .
            calculer  $\chi_{[K]} = \sum_{i=0}^3 |\deg[p_i] - d_i^*|^2$ .
        pour chaque maille  $K_j$  faire
            vérifier que  $\det[J_{K_j}] \geq \epsilon$  et que  $q[K_j] \geq q_{\min}$ .           ▷ non dégénérescence.
            vérifier que  $\langle \mathbf{n}[p_{ij}], \mathbf{n}[K_j] \rangle \in [0, \cos(\theta_{\max})], \forall p_{ij} \in K_j$ .
            vérifier que l'aire du projeté de  $K_j$  sur  $T_{p_{ij}}\Gamma$  est positive,  $\forall p_{ij} \in K_j$ .
        fin
        si tout est ok et  $\chi_{[K]} < \chi_{[M]}$  alors
            appliquer la bascule de c de manière persistante sur  $\mathcal{T}_h$ .
            passer à la maille suivante.
        fin si
    fin si
    démarquer c.
fin
fin
jusqu'à ce qu'aucune bascule possible ou  $t \geq t_{\max}$ 
```

## Algorithme 3.5: Lissage

**assure que :**

- la qualité de la pire maille incidente à chaque point est améliorée.
- la discréttisation des courbes de discontinuité est régulière.
- la contrainte de continuité  $G^1$  est respectée.

**1. Lissage des courbes de discontinuité.**

**pour** chaque "ridge" identifié par  $uid = M[p]$  **faire**  $\triangleright uid \in (R, S, \beta)$ .

- extraire  $[r, \tilde{r}, q, \tilde{q}] = [R[\beta[uid]^-], R[\beta[uid]^+]]$ , leurs normales ainsi que leurs densités.  $\triangleright I = \{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\}$ .
- construire la B-spline quartique  $\gamma : I \rightarrow \Gamma$  passant par  $r, p, q$ .  $\triangleright \omega_i = \|\dot{\gamma}[t_i]\| \rho[\gamma[t_i]]$ .
- calculer le paramètre  $s = [\sum_{t_i \in I} \omega_i t_i][\sum_{t_i \in I} \omega_i]^{-1}$ .  $\triangleright [p, \tilde{p}] = R[uid]$ .
- sauvegarder  $p_{old} = p$  et remplacer  $p$  et son jumeau  $\tilde{p}$  par  $\gamma[s]$ .
- pour** chaque maille  $K$  incidente à  $p$  ou  $\tilde{p}$  **faire**

  - si**  $\det[J_K] < \epsilon$  ou  $q[K] < q_{min}$  **alors**  $\triangleright$  dégénérescence.
  - remettre  $p = \tilde{p} = p_{old}$  et sortie de boucle.  $\triangleright$  cas très rare.

- fin**

**fin**

**2. Lissage des régions internes.****pour** chaque point régulier  $p$  **faire**DIFFUSION.réinitialiser  $t = \vec{0}$  et  $\|S\| = 0$ , et évaluer  $|q|_0 = \min q[K]$ .**pour** chaque maille  $K$  incidente à  $p$  **faire**calculer le patch quartique  $\Gamma_K$  associé à  $K$ .

**pour** chaque point de quadrature  $c_i$  de  $K$  **faire**  $\triangleright c_i = \gamma_i(\frac{1}{2})$  pour tout  $\gamma_i \in \partial\Gamma_K$ .

- interpoler la densité  $\rho[c_i]$  et la jacobienne  $J_{c_i}$ .
- calculer  $\omega_i = \rho[c_i] \det[J_{c_i}^\top J_{c_i}]$  et accumuler  $\|S\| = \|S\| + \omega_i$ .
- ajuster la direction  $t = t + \omega_i \log_p[c_i]$ .  $\triangleright$  projection sur  $T_p\Gamma$ .

**fin****fin**  $\triangleright t = \int_S x \rho[x] dx - p$ .ajuster  $t = \frac{1}{\|S\|}t$ .trouver la maille  $\tilde{K}$  telle que  $\exp_p[\alpha_i t] \in \Gamma_{\tilde{K}}$ .  $\triangleright$  un  $\alpha_i \approx \frac{1}{4}$  suffit.prendre  $\alpha = \max_i \alpha_i$ .**répéter**prendre  $p = \exp_p[\alpha t]$  et réévaluer la normale  $n[p]$  et la densité  $\rho[p]$ .  $\triangleright$  projection.**pour** chaque maille  $K$  incidente à  $p$  **faire**vérifier que  $\det[J_K] \geq \epsilon$  et que  $q[K] \geq |\varphi|_0$ .  $\triangleright$  non dégénérescence.vérifier que  $\langle n[p_j], n[K] \rangle \in [0, \cos(\theta_{max})]$ ,  $\forall p_j \in K$ .  $\triangleright$  inversion ou déviation.projeter  $K$  sur  $T_{p_j}\Gamma$ , et vérifier que  $\det[J_{\tilde{K}}] \geq \epsilon$ ,  $\forall p_j \in K$ .  $\triangleright$  distorsion sur  $T_{p_j}\Gamma$ .**si** l'un des critères n'est pas ok **alors**diminuer  $\alpha = \frac{\alpha}{2}$  et sortir de boucle.  $\triangleright$  relaxation.**fin****jusqu'à** ce que toutes les mailles soient ok ou  $\alpha \leq \epsilon$ .DESCENTE.**si** déplacement non accepté **alors**  $\triangleright$  on se place dans un repère local de  $T_p\Gamma$ .calculer  $f_i[p] = -\min q[K_i]$  et  $t^{[0]} = -\nabla f_i[p]$ .  $\triangleright$  s'obtient en perturbant légèrement  $p$ .calculer le pas initial  $\alpha^{[0]}$  à l'aide de (3.18).  $\triangleright$  tenir compte des autres mailles.initialiser  $t = 0$ .**répéter**calculer  $p^{[t]} = \exp_p[\alpha^{[t]} t^{[t]}]$ .  $\triangleright$  déplacement puis projection.réévaluer  $t^{[t]}$  et incrémenter  $t$ .  $\triangleright$  perturber légèrement  $p$ .calculer le pas  $\alpha^{[t]}$  par recherche linéaire (voir page 73).  $\triangleright$  critères de Wolfe.**jusqu'à**  $d_t p^{[t]} \leq \epsilon$  ou  $t \geq t_{max}$ .  $\triangleright$  déplacement significatif.vérifier que  $\forall p_j \in K$ ,  $\langle n[p_j], n[K] \rangle \in [0, \cos(\theta_{max})]$ , si ok : accepter le déplacement.**fin si****fin**

### 3.2.5 Stratégie complète

À ce point, nous disposons de quatre noyaux dédiés au rééchantillonnage (**raffinement**, **simplification**) et à la régularisation (**relaxation**, **lissage**) de la surface triangulée, et telle que les coins et les courbes saillantes de la surface soient préservées. Ainsi, il nous manque :

- une **carte de densité** décrivant la répartition souhaitée de points sur chaque région de la surface. Notons qu'elle est relative à l'erreur estimée de la surface ou bien celle d'une solution numérique calculée sur la triangulation (voir figure 3.1). De manière concrète, nous construisons un champ de tenseurs métriques pour encoder les prescriptions de tailles à la fois dans le cas isotrope et anisotrope. Ainsi sa construction est expliquée dans le cadre général à la section 3.3.1.
- la carte de densité précédente est extraite des **courbures locales** de la surface dans le premier cas, et à partir des **hessiennes locales** de la solution numérique dans le second cas. Il faudrait donc pouvoir reconstruire ces quantités sur la surface discrète. Afin de ne pas alourdir le chapitre, les méthodes utilisées sont décrites en annexe aux pages 137 et 134.

Muni de ces ingrédients, nous pouvons à présent mettre en place notre remailleur adaptatif, dont le principe est résumé à la figure 3.17 et les étapes illustrées à la figure 3.18.

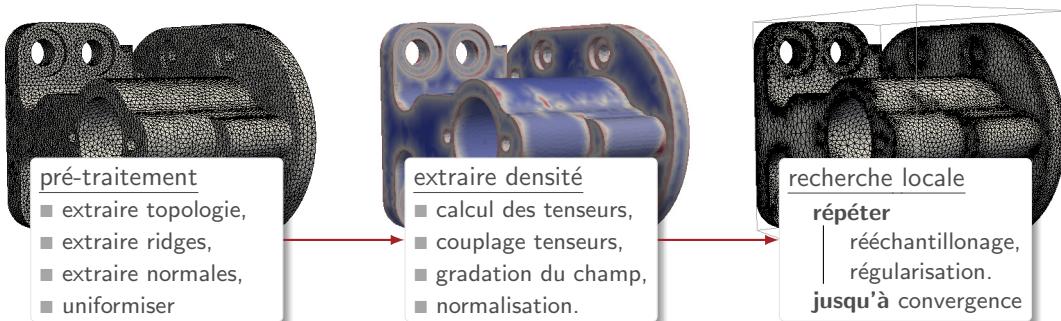


Figure 3.17: Principe du remailleur adaptatif.

PARAMÈTRES. Rappelons que notre but est de fournir la "meilleure" surface triangulée qui soit adaptée à l'erreur estimée de la surface ou d'une solution numérique, étant donné un budget de points (voir page 54). Ainsi nous disposons d'une triangulation initiale satisfaisant les critères décrits à la page 56 en entrée, avec quatre paramètres additionnels :

- $n_{\max}$  : c'est la résolution souhaitée, i.e le nombre de points de la surface triangulée.
- $\theta_{\max}$  : c'est la déviation maximale des mailles par rapport au plan tangent de chaque point<sup>59</sup>.
- $h_{\max}$  : c'est le diamètre autorisé<sup>60</sup>, et qui permet de borner les tailles d'arêtes dans les régions à courbure nulle de la surface<sup>61</sup>.
- $h_{\text{grad}}$  : c'est le seuil de gradation i.e le ratio maximal de longueurs de paires d'arêtes incidentes.

ALGORITHME. In fine, le remailleur est constitué d'une séquence de trois phases : le pré-traitement, la construction de la métrique pour la répartition des points et la boucle de recherche locale pour les modifications proprement dites sur la surface triangulée.

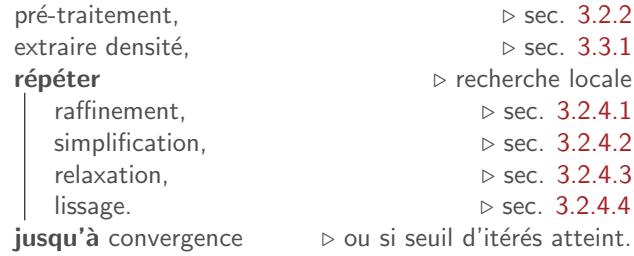
Notons juste que trouver la triangulation optimale qui minimise une fonctionnelle donnée est NP-difficile. En fait tous les remailleurs existants sont soit des heuristiques (gloutonnes ou recherche locale) soit au mieux des algorithmes d'approximation (voir section 2.2, page 42). Ainsi aucune combinaison particulière de noyaux n'a été formellement prouvée comme étant optimale. Ici, nous ne sommes pas réellement intéressés par trouver la combinaison la plus efficiente. Ainsi nous utilisons une simple boucle de recherche locale décrite à l'algorithme 3.6 pour nos évaluations numériques.

<sup>59</sup>Autrement dit, c'est le seuil angulaire des normales aux mailles et à leurs sommets respectifs.

<sup>60</sup>Le diamètre d'un maillage est la longueur de sa plus grande arête

<sup>61</sup>Dans notre cas, il n'est pas nécessaire de prescrire une taille minimale.

**Algorithme 3.6: Remaillage adaptatif**



- PRÉ-TRAITEMENT. Elle consiste à extraire les données requises par les noyaux et à uniformiser la triangulation initiale si nécessaire. Les "ridges" et "coins" sont d'abord identifiés et extraits. Ensuite la topologie est explicitement construite et stockée sous forme d'un graphe  $(P, M, \mathcal{N})$  pour la connectivité des points et mailles d'une part, et d'un graphe  $(R, S, \beta)$  pour le chainage des arêtes vives, ce qui est utile pour la reconstruction des courbes saillantes de la surface. La surface discrète est ensuite orientée grâce à un champ de vecteurs normaux permettant de distinguer son intérieur et son extérieur. Enfin, si cela est nécessaire, la triangulation est régularisée en vue d'obtenir une répartition quasi-uniforme des points et de stabiliser les étapes ultérieures<sup>62</sup>.
- EXTRAIRE DENSITÉ. Elle consiste à définir un champ décrivant la répartition souhaitée des points sur la surface. De manière concrète, un champ de tenseurs métriques est construit à partir de l'erreur estimée de la surface ou bien d'une solution numérique  $\mathbf{u}$  calculée sur la triangulation. Dans le premier cas, le tenseur métrique associé à un point  $p$  est calculé à partir du tenseur de courbure en  $p$ . Dans le second cas, il est extrait à partir de la hessienne locale de  $\mathbf{u}$  en  $p$ . Plus précisément, elle se fait en trois passes :
  - la reconstruction des courbures et hessiennes locales de  $\mathbf{u}$  (voir annexe, page 137).
  - la gradation du champ afin de lisser les variations de tailles de toute paire d'arêtes incidentes.
  - la normalisation du champ pour mieux répartir les points selon la résolution souhaitée  $n_{\max}$ .
- RECHERCHE LOCALE. Elle consiste à construire la suite de triangulations de sorte à converger vers la discrétisation "optimale". Pour cela, une boucle entrelaçant les étapes de rééchantillonnage (raffinement, simplification), et de régularisation (relaxation et lissage) est exécutée tant que l'erreur décroît ou que la qualité des mailles croît de manière significative.
  - rééchantillonnage. Ici le but est d'insérer ou de supprimer les points afin d'obtenir des arêtes de taille quasi-unité dans l'espace métrique induit par la densité<sup>63</sup>. Les points créés sont directement projetés sur la surface. La suppression de points est entre autres conditionnée par un contrôle sur la déviation des mailles aux plans tangents de leurs sommets. Enfin, le nombre de points est contrôlé lors d'une simplification par une priorisation des tâches<sup>64</sup>.
  - régularisation. La surface discrète est ensuite régularisée en termes de degré des points et de qualité des mailles. Pour cela, la connectivité et la position des points sont modifiées par le biais de bascules d'arêtes et de lissage, en vue d'améliorer l'échantillonnage précédent. Là encore, un contrôle sur la déviation des mailles est effectué afin d'assurer la continuité  $G^1$ .

<sup>62</sup>Le maillage fourni en entrée devrait idéalement être uniforme, ou du moins une assez bonne discrétisation de la surface idéale. Comme l'adaptation vient après une phase de calcul dans la boucle adaptative, on s'attend à ce que le maillage initial fourni au solveur soit de qualité acceptable. Si ce n'est néanmoins pas le cas, les aires des mailles peuvent être égalisées par une boucle entrelaçant les passes de lissage et de bascule d'arêtes basée sur le critère de DELAUNAY. Rappelons juste que le but de l'adaptation n'est pas d'améliorer la qualité des mailles, mais de fournir une discrétisation qui minimise une erreur tout en préservant la qualité des mailles.

<sup>63</sup>Plus précisément, le but est d'obtenir un maillage uniforme unité de la variété riemannienne induite par le champ de tenseurs métriques.

<sup>64</sup>basée sur le cumul de l'erreur locale en chaque point.

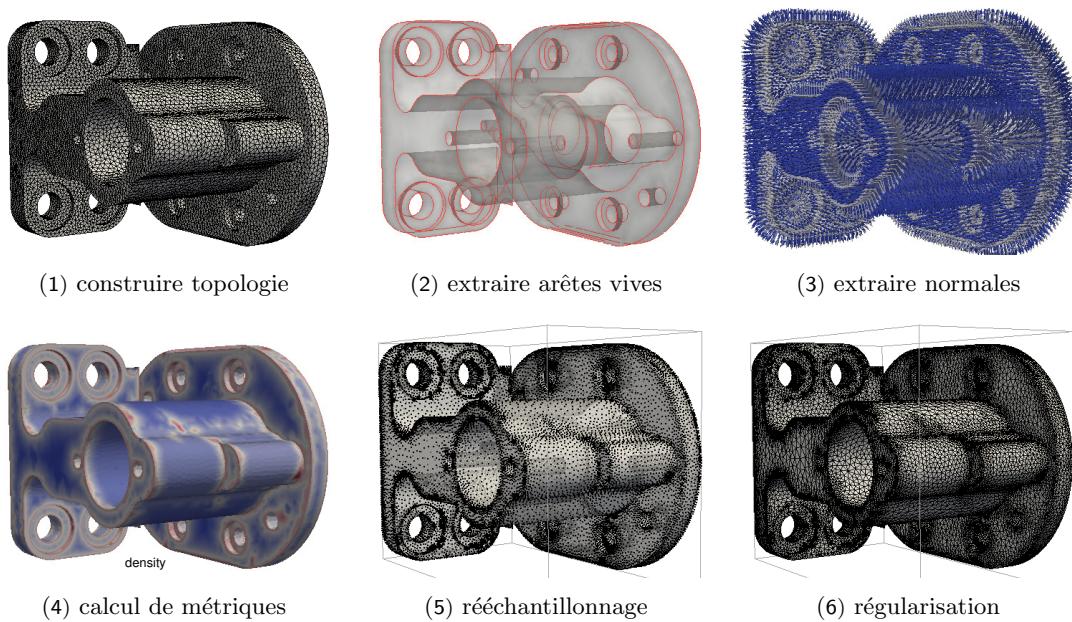


Figure 3.18: Illustration synthétique des étapes de l'algorithme.

### 3.3 EXTENSION AU CAS ANISOTROPE

#### 3.3.1 Répartition anisotrope des points

##### 3.3.1.1 Intérêt et principe

INTÉRÊT. À ce point, nous avons défini les noyaux ainsi qu'une stratégie complète pour l'adaptation locale de surfaces triangulées. Notons que ces noyaux dépendent d'une densité qui définit la répartition souhaitée des points sur la surface. Néanmoins nous n'avons pas encore construit explicitement cette densité. En fait, elle peut être **isotrope** ou **anisotrope** selon que la taille souhaitée d'une arête  $[pq]$  incidente à un point  $p$  dépend de la direction de  $\vec{pq}$  ou non (voir figure 2.12, page 47). Le premier cas est plus simple à gérer puisque la densité se réduit à un scalaire sur chaque point. Le second cas est plus complexe puisqu'il faut encoder la densité dans toutes les directions incidentes à chaque point.

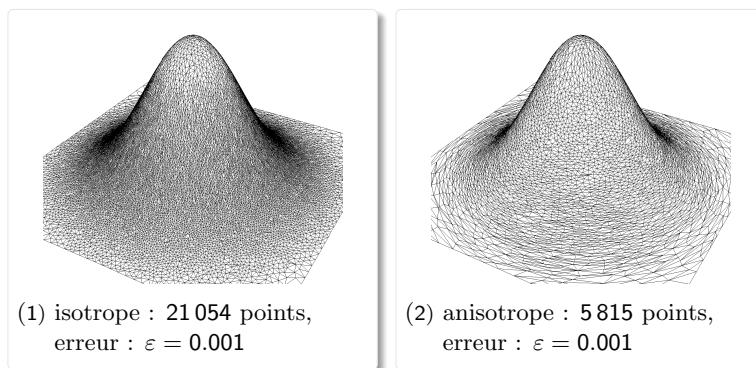


Figure 3.19: Intérêt d'une triangulation anisotrope comparée à sa version isotrope.

En pratique, une triangulation anisotrope permet de réduire significativement le nombre de points requis comparé à sa version isotrope pour un même seuil d'erreur comme illustré à la figure 3.19. Par ailleurs, rappelons nous qu'une triangulation "idéale" est celle dont la densité des points, ainsi que la forme et l'étirement des mailles sont adaptées à l'erreur estimée de la solution numérique ou de la surface elle-même (voir page 44). Ainsi il est crucial de capturer l'anisotropie de cette erreur.

**PRINCIPE.** Afin de prescrire la répartition souhaitée de points, nous assignons un **tenseur métrique** en tout point de la surface (voir page 50). L'avantage est qu'il n'est pas nécessaire de modifier les noyaux si on souhaite adapter la triangulation à un autre critère, contrairement à [131] par exemple. En effet seule la manière de calculer les distances change. Ainsi cela nous permet d'utiliser une structure unique pour encoder aussi bien l'erreur de la solution numérique que celle de la surface en vue d'inférer la densité souhaitée sur chaque point.

Rappelons nous que le tenseur métrique  $g_p$  relatif à un point  $p$  redéfinit son voisinage continu comme montré à la figure 2.12 (page 47). Ainsi cela permet de prescrire une taille d'arête qui tienne compte de la direction de  $\vec{pq}$  pour toute arête  $[pq]$ . De manière concrète, le tenseur est représenté par une matrice symétrique définie positive dont les valeurs propres encodent les tailles d'arêtes souhaitées en direction des vecteurs propres  $\mathbf{v}_i$ <sup>65</sup>. Dans une base du plan tangent de  $p$ , il s'écrit :

$$g_p = \mathbf{P}^T \mathbf{D} \mathbf{P}, \text{ avec } \begin{cases} \mathbf{D} = \text{diag}(h_{i,p}^{-2})_{i=1}^2 : \text{les valeurs propres de } g_p, \\ \mathbf{P} = (\mathbf{v}_1, \mathbf{v}_2) : \text{les directions principales de } g_p, \\ h_{i,p} : \text{la taille d'arête prescrite en direction de } \mathbf{v}_i. \end{cases} \quad (3.22)$$

En fait, on veut construire un champ tensoriel<sup>66</sup> qui permet de capturer l'erreur de la surface ou celle d'une solution numérique  $\mathbf{u}$  en vue de guider les noyaux (voir figure 3.20). Intuitivement,

- l'erreur d'approximation de la surface est plus importante dans les régions où elle varie le plus.
- l'erreur d'interpolation de  $\mathbf{u}$  est plus importante dans les régions où son gradient varie le plus.

Ainsi ce champ est construit à partir des **courbures locales** de la surface ou des **hessiennes locales**<sup>67</sup> de  $\mathbf{u}$  sur chaque point. En fait, ces deux tenseurs (courbure, hessienne) doivent être reconstruits de manière discrète sur chaque point. Dans le cas où  $\mathbf{u}$  est fournie en entrée, ces tenseurs seront couplés de manière à avoir un tenseur unique par point, et qui permet de guider les noyaux de manière à minimiser ces deux erreurs. Afin de ne pas alourdir la section, les méthodes afférentes sont décrites en annexe (page 137).

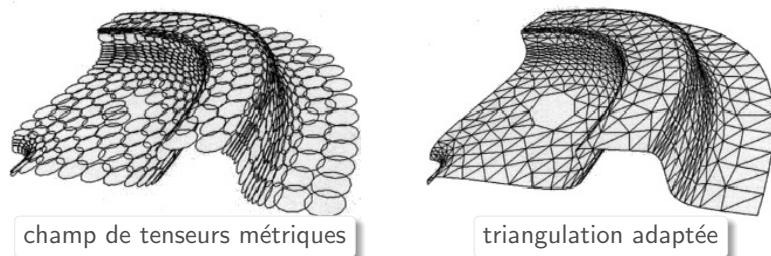


Figure 3.20: Principe de l'adaptation guidée par une métrique riemannienne.

**CONSTRUCTION.** Le tenseur précédent encode les informations relatives à la densité souhaitée au voisinage de chaque point. Néanmoins il n'est pas encore utilisable en l'état car pour cela il faudrait incorporer les contraintes sur le seuil d'erreur ou le nombre cible de points. Pour obtenir un tenseur métrique valide, il faudrait utiliser un estimateur d'erreur. En fait, il permet de normaliser le champ tensoriel en fonction du nombre cible de points  $n_{\max}$  d'une part, ainsi qu'à la sensibilité de la norme utilisée pour capturer l'erreur d'autre part. Elle sera décrite à la section 3.3.1.2. Enfin, bien cette normalisation permette de répartir équitablement les points de  $\Gamma$ , elle peut éventuellement induire des variations abruptes de taille entre deux points voisins. Ainsi une gradation est nécessaire afin de garantir une variation lisse de tailles d'arêtes incidentes. Elle sera détaillée à la section 3.3.1.3.

<sup>65</sup>Notons que les vecteurs propres dépendent de la base choisie (locale ou canonique).

<sup>66</sup>ou **métrique riemannienne**

<sup>67</sup>Intuitivement, elle représente la courbure intrinsèque de  $\mathbf{u}$ . Formellement, il s'agit d'une matrice symétrique définie positive dont les coefficients sont relatifs aux dérivées seconde de  $\mathbf{u}$ .

### 3.3.1.2 Normalisation

PRINCIPE. À vrai dire, adapter la triangulation à un champ de métriques revient à générer une triangulation uniforme d'une variété riemannienne (définition 13, page 24). En effet on veut que toute arête soit de longueur unité dans cet espace métrique. En fait c'est le principe de la plupart des remailleur surfaciques adaptatifs comme YAMS, MMGS, ou encore MADLIB [108, 96, 132]. Notons néanmoins qu'ils sont basés sur un contrôle explicite de l'erreur<sup>68</sup> : ils visent à trouver la triangulation qui respecte ce seuil d'erreur avec le moins de points possibles. Dans notre cas, c'est l'inverse.

Mais alors, comment inférer une métrique relative à cet erreur quand on ne peut pas la borner ?

Pour cela, on recourt à un estimateur d'erreur basé sur la notion de **maillage continu**. Il permet d'estimer l'erreur d'interpolation d'une fonction  $f \in C^2(\Omega, \mathbb{R})$  sur un plan ou un volume  $\Omega \subset \mathbb{R}^d$ , et de trouver un champ tensoriel qui minimise cet erreur dans le cadre continu par le biais d'un calcul variationnel [75, 98]. Ici, l'idée est de trouver un même champ permettant de contrôler aussi bien l'erreur d'une solution numérique  $\mathbf{u}$  que celle de la surface. Ainsi l'avantage est double, car il permet :

- d'inférer la répartition de l'erreur directement en fonction du nombre cible de points,
- de définir une densité qui capture les variations d'échelles différentes de cet erreur, en ajustant la sensibilité de la norme  $L^p$  utilisée, où  $p$  est un paramètre à fixer.

DÉTAILS. En fait, notre cas est particulier car on n'est ni en planaire ni réellement en 3D non plus. En effet la surface représente la frontière d'un domaine volumique, et n'est que "plongée" dans  $\mathbb{R}^3$ . Mais comme il s'agit d'une variété, chaque tenseur peut s'exprimer dans une base locale du plan tangent  $T_p\Gamma$  du point  $p$ . Il est donc localement de rang deux. Ainsi il peut s'exprimer comme une matrice de  $\mathbb{R}^{2 \times 2}$  dans la base locale de  $T_p\Gamma$ . Dans notre cas, il s'écrit :

$$\forall \mathbf{u}, \mathbf{v} \in T_p\Gamma, g_p(\mathbf{u}, \mathbf{v}) = \langle \mathbf{u}, J_p(\alpha \mu_p M_p) J_p^\top \mathbf{v} \rangle \quad (3.23)$$

où :

- $J_p \in \mathbb{R}^{3 \times 2}$  est la matrice jacobienne de la paramétrisation locale de la surface au point  $p$ . Elle permet de passer du repère local au plan tangent  $T_p\Gamma$  de  $p$  à la base canonique de  $\mathbb{R}^3$ .
- $M_p \in \mathbb{R}^{2 \times 2}$  est la matrice décrite en (3.22) et qui exprime la densité prescrite sur  $p$  dans les directions principales de son plan tangent  $T_p\Gamma$ . Elle provient du tenseur de courbure, ou de la hessienne de la solution numérique exprimés dans la base locale de  $T_p\Gamma$ , ou de leur intersection.
- $\mu_p \in \mathbb{R}$  est un facteur qui permet de contrôler la sensibilité de l'estimateur d'erreur dans les régions où la solution numérique ou la surface varie fortement, en fonction de la norme  $L^p$  choisie.

$$\mu_p = (\lambda_{1,p} \lambda_{2,p})^{-1/(2p+2)}. \quad (3.24)$$

- $\alpha \in \mathbb{R}$  est un facteur qui permet d'harmoniser la répartition souhaitée des points, exprimée par la complexité  $C_h$  du champ tensoriel, en fonction de la résolution  $n_{\max}$ .

$$\alpha = \frac{n_{\max}}{C_h}, \text{ avec } C_h = \int_{S=\Gamma} \rho[x]^{2p/(2p+2)} dx = \sum_{K \in \mathcal{T}_h} \int_K [\lambda_{1,x} \lambda_{2,x}]^{2p/(2p+2)} dx. \quad (3.25)$$

Les exposants impliqués dans les expressions de  $\mu_x$  et  $C_h$  sont équivalents à ceux de [75] mais adapté au cas surfacique.

STOCKAGE. En pratique, les tenseurs sont stockés de manière discrète sur les points de la triangulation. En fait, le tenseur associé à un point  $p$  est en réalité une matrice de  $\mathbb{R}^{2 \times 2}$  une fois la base locale de son plan tangent  $T_p\Gamma$  fixée, voir (3.23). Afin de minimiser le coût mémoire, on aurait pu se contenter de ne stocker qu'une matrice de  $\mathbb{R}^{2 \times 2}$ . Néanmoins, cela n'est possible que si on dispose d'une paramétrisation globale sur la surface. Ainsi on est donc obligé de stocker en plus la matrice

<sup>68</sup>Dans MMGS, un estimateur d'erreur en norme infinie est utilisé afin d'approcher la distance de Hausdorff entre la surface idéale et discrète. Dans yams, c'est plutôt un contrôle sur l'erreur relative à chaque arête qui est effectué. Dans les deux cas, cela permet d'inférer directement une carte de tailles de sorte que l'erreur n'excède pas un seuil  $\bar{\varepsilon}$  fixé.

jacobienne<sup>69</sup> associée à  $p$ , ce qui fait en tout  $6 + 3$  coefficients par point. Ainsi nous préférons exprimer directement chaque tenseur<sup>70</sup> comme une matrice de  $\mathbb{R}^{3 \times 3}$  de sorte que  $g_p(\mathbf{v}, \mathbf{n}[p]) = 0$  pour tout  $\mathbf{v} \in \mathbb{R}^3$ . Enfin, puisqu'il s'agit d'une matrice symétrique, on ne stocke que sa partie triangulaire supérieure en mémoire. La routine complète relative à la construction et à la normalisation de la métrique est donné à l'algorithme 3.7.

Algorithme 3.7: Construction et normalisation de la métrique.

**1. Couplage des tenseurs**

```

pour chaque point  $p$  faire en parallèle           ▷ réduction dans une base commune  $P$ .
    poser  $g_p = 0_{3 \times 3}$ .
    calculer une base locale  $J = (\partial_u p, \partial_v p)$  du plan tangent de  $p$ .
    calculer  $M_1$  le tenseur de courbure de  $p$ .
    calculer  $M_2$  la hessienne de  $u$  sur  $p$  dans la base  $J$ .           ▷  $M_2 = J^T H_{u,p} J$ .
    calculer  $N = M_1^{-1} M_2$  et diagonaliser en  $R^{-1} D R$ .           ▷ voir équation A.16.
    pour chaque direction  $\mathbf{v}_i \in \mathbb{R}$  faire
        
$$\begin{cases} \lambda_{1,i} = \langle \mathbf{v}_i, M_1 \mathbf{v}_i \rangle. \\ \lambda_{2,i} = \langle \mathbf{v}_i, M_2 \mathbf{v}_i \rangle. \\ D_{ii} = \min(\max(|\lambda_{1,i}|, |\lambda_{2,i}|), h_{\max}^{-2}). \end{cases}$$
           ▷ voir équation A.17.
    fin
    stocker  $g_p = J P^T D P J^T$  avec  $P = R^{-1}$ .           ▷ voir équation A.18.
barrière

```

**2. Normalisation selon la sensibilité de la norme  $L^p$ .**

```

pour chaque point  $p$  faire en parallèle
    calculer une base locale  $J = (\partial_u p, \partial_v p)$  du plan tangent de  $p$ .
    calculer  $M$  le tenseur associé à  $g_p$  dans la base  $J$ .
    diagonaliser  $M$  en  $R^{-1} D R$ .
    pour chaque valeur propre  $D_{ii}$  faire
        
$$\begin{cases} D_{ii} = \max(\epsilon_{\text{num}}, |D_{ii}|). \\ \mu = [\det D]^{-1/(2p+2)}. \end{cases}$$
           ▷  $\epsilon_{\text{num}} = o(10^{-14})$ .
    stocker  $g_p = J R^{-1} \mu D R J^T$ .
barrière

```

**3. Normalisation selon le nombre cible de points  $n_{\max}$ .**

```

    poser  $C_h = 0$ .           ▷ calcul de la complexité du champ.
    pour chaque maille  $K$  réduire  $C_h$  en parallèle           ▷ voir équation 3.25.
        poser  $\alpha = 0$ 
        pour chaque point  $p$  de  $K$  faire
            accumuler  $\alpha = \alpha + \det(g_p)^{-2p}$ .           ▷  $\det(g_p) = [\lambda_{1,p} \lambda_{2,p}]^{-1/(2p+2)}$ .
            accumuler  $C_h = C_h + |K| \alpha$ .           ▷  $C_h = \sum_{K \in \mathcal{T}_h} \frac{|K|}{3} \sum_{i=1}^3 \det(g_{x_i})^{-2p}$ .
        barrière
        poser  $\lambda = (3 n_{\max} / C_h)$ .           ▷ voir équation 3.25.
        pour chaque point  $p$  faire en parallèle
            calculer une base locale  $J = (\partial_u p, \partial_v p)$  du plan tangent de  $p$ .
            calculer et diagonaliser  $M = R^{-1} D R$  le tenseur associé à  $g_p$  dans la base  $J$ .
            stocker  $g_p = J R^{-1} \lambda D R J^T$ .           ▷ voir équation 3.23,  $D = \mu D$ .
        barrière

```

<sup>69</sup>Elle permet de passer du repère local du plan tangent à la base canonique de  $\mathbb{R}^3$ .

<sup>70</sup>Notons juste que le tenseur est congruent à une matrice diagonale dans une base de  $\mathbb{R}^3$  formé par les composantes de  $J_p$  et la normale  $\mathbf{n}[p]$ .

### 3.3.1.3 Gradation

Le champ tensoriel obtenu à la section précédente décrit la répartition idéale des points en fonction de l'approximation de la surface ou de l'interpolation de la solution numérique  $\mathbf{u}$ , selon le nombre cible de points  $n_{\max}$  et la sensibilité de la norme  $L^p$  utilisée. Néanmoins il ne garantit pas nécessairement une répartition régulière des points sur toute la surface. Ainsi cela peut induire un ratio important de tailles d'arêtes voisines impliquant des mailles très étirées et de piètre qualité. Afin d'assurer une variation graduelle de tailles d'arêtes sur la surface triangulée, nous procédons au lissage du champ tensoriel : c'est ce que nous qualifions de gradation dans notre contexte.

TRAVAUX CONNEXES. Rappelons que la taille cible d'arête au point  $p$  en direction de  $\mathbf{u}$  est  $h_p(\mathbf{u}) = g_p(\mathbf{u}, \mathbf{u})^{-\frac{1}{2}}$ . Il y a plusieurs manières de considérer le problème selon que l'on veuille lisser (ou borner) :

- size ratio, i.e le ratio de tailles d'arêtes adjacentes [132, 99].
- H-variation, i.e le gradient de  $h$  en direction de  $\vec{pq}$  [100].
- H-shock, i.e le ratio de  $h$  relative à la variation de  $\ell_h(pq)$  [100, 101].

Ainsi le but est donc de mettre à jour  $g_p$  en fonction d'un seuil sur l'un des critères ci-dessus. Notons indistinctement ce critère  $\mathbf{c}_p(\mathbf{u})$  pour un point  $p$  donné en direction de  $\mathbf{u} = \vec{pq}$ . En fait il n'existe pas mille façons d'y parvenir : étant donné un seuil  $\epsilon$  sur  $\mathbf{c}_p$ , il faut trouver un facteur d'atténuation  $\lambda_i$  à appliquer sur chaque  $h_p(\mathbf{v}_i)$  pour chaque direction principale  $\mathbf{v}_i$  dans un premier temps. On réitère la routine jusqu'à ce que  $\mathbf{c}_p$  soit atteint pour tout  $p$ .

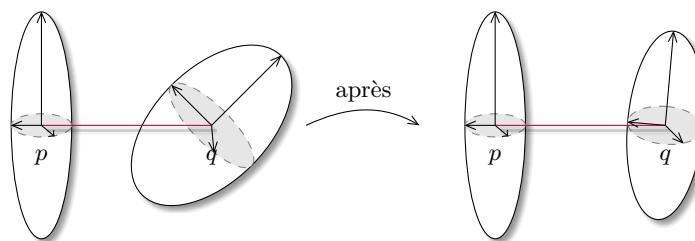


Figure 3.21: Nécessité d'un ajustement directionnel des tenseurs.

À vrai dire, les approches ne se distinguent réellement que sur la manière d'ajuster les directions principales  $\mathbf{v}_i$  associées à chaque  $h_p(\mathbf{v}_i)$  dans le cas anisotrope (voir figure 3.21). En effet, un lissage des directions est nécessaire quand l'orientation des tenseurs relatifs aux sommets d'une arête  $[pq]$  sont trop éloignés, afin d'obtenir un alignement régulier des mailles incidentes à  $p$  et  $q$ . Ainsi les directions principales  $\mathbf{v}_i$  peuvent être :

- gardées intactes comme dans [132]. Ainsi la gradation se résume en l'application d'un facteur d'homothétie sur chaque  $h_x(\mathbf{v}_i)$  pour chaque direction principale  $\mathbf{v}_i$  en tout point  $x$ .
- réduites par intersection de tenseurs comme dans [101]. La matrice de direction  $R$  du tenseur à modifier  $M = R \Lambda R^T$  est ajusté par rotation d'angle  $\theta$ , de sorte que les deux tenseurs soient congruents à une matrice diagonale dans la base formée par  $R$ . Il rend le tenseur isotrope dans le cas où les orientations des deux tenseurs sont trop éloignées comme sur la figure A.3.
- ajustées au cas par cas comme dans [99] selon la configuration des directions principales associées à chaque paire de tenseurs, qui peuvent être :
  - de rang 1, avec une direction unique (sphérique).
  - de rang 2, avec une direction polaire et une équatoriale (sphéroïdal).
  - de rang 3, avec trois directions distinctes (ellipsoïdal).

Elle préserve mieux l'anisotropie, mais la matrice de direction ne reste pas toujours orthogonale.

En notant  $S$  le voisinage de  $p$ , les trois critères de gradation décrits précédemment sont donnés par :

$$\mathbf{s}(p) = \max_{q,r \in S} \frac{\|pq\|}{\|pr\|} \quad (3.26a)$$

$$\nabla h_p = \frac{h_q - h_p}{\|pq\|} \quad (3.26b)$$

$$\mathbf{c}(pq) = \max \left( \frac{h_p}{h_q}, \frac{h_q}{h_p} \right)^{1/\ell_h(pq)} \quad (3.26c)$$

En réalité, le choix du critère n'est pas décisif puisque les trois critères sont équivalents (propriété 2).

**Propriété 2** (ÉQUIVALENCE DES MESURES DE GRADATION). *Les trois mesures de gradation (size ratio, H-variation et H-shock) décrits à (3.26) sont équivalentes.*

PREUVE. Soient  $\epsilon_1$ ,  $\epsilon_2$  et  $\epsilon_3$  les seuils prescrits pour chaque critère de l'équation (3.26). Pour cela nous allons démontrer que borner  $\nabla h_p$  par  $\epsilon_2$  est équivalent à borner  $\mathbf{s}(p)$  dans un premier temps. Ensuite, nous montrons que borner  $\mathbf{s}(p)$  par  $\epsilon_1$  est équivalent à borner  $\mathbf{c}(pq)$  pour tout  $q \in S$ .

En supposant que  $h_q \geq h_p$ , on a :

$$\nabla h_p = \epsilon_2 \quad (3.27a)$$

$$\Leftrightarrow \frac{h_q - h_p}{\|pq\|} = \epsilon_2 \quad (3.27b)$$

$$\Leftrightarrow h_q - h_p = \epsilon_2 \|pq\| \quad (3.27c)$$

$$\Leftrightarrow h_q - h_p = \epsilon_2 \ell_p(pq) h_p \quad (3.27d)$$

$$\Leftrightarrow h_q = h_p(\epsilon_2 \ell_p(pq) + 1) \quad (3.27e)$$

$$\Leftrightarrow \boxed{\frac{h_q}{h_p} = 1 + \epsilon_2 \ell_p(pq)} \quad (3.27f)$$

D'un autre côté, on va exprimer le lien entre le variation de  $h$  et le ratio de taille  $\mathbf{s}(p)$ . On suppose qu'on dispose d'une triangulation unité au sens où les tailles d'arêtes sont conformes avec la métrique  $g_x$  pour tout  $x$ . Dans ce cas, pour toute paire d'arête incidente  $[ab]$  et  $[bc]$ , on a :  $\ell_h(ab) = \ell_h(bc) = 1$ . Ici l'idée est d'exprimer  $\|ab\|$  et  $\|bc\|$  en fonction de la longueur de la courbe  $\gamma$  sous-tendue par  $[ac]$ . En notant  $\hat{a}$ ,  $\hat{b}$  et  $\hat{c}$  les coordonnées paramétriques de  $a$ ,  $b$  et  $c$  sur  $\gamma : [0, 1] \rightarrow \mathbb{R}$  avec  $\hat{a} = 0$  et  $\hat{c} = 1$ . On a :

$$1 = \ell_h(ab) \quad (3.28a)$$

$$\Leftrightarrow 1 = \int_{\hat{a}}^{\hat{b}} \frac{|\gamma'(t)|}{h(\gamma(t))} dt \quad (3.28b)$$

$$\Leftrightarrow 1 = \ell(\gamma) \int_{\hat{a}}^{\hat{b}} \frac{1}{h_a + (h_c - h_a)t} dt \quad (3.28c)$$

$$\Leftrightarrow 1 = \frac{\ell(\gamma)}{h_c - h_a} \int_{\hat{a}}^{\hat{b}} \frac{h_c - h_a}{h_a + (h_c - h_a)t} dt \quad (3.28d)$$

$$\Leftrightarrow 1 = \frac{1}{\nabla h} (\ln |(h_c - h_a)\hat{b} + h_a| - \ln |(h_c - h_a) + h_a|) \quad (3.28e)$$

$$\Leftrightarrow \nabla h = \ln \left| \frac{(h_c - h_a)\hat{b} + h_a}{h_a} \right| \quad (3.28f)$$

$$\Leftrightarrow e^{\nabla h} - 1 = \frac{(h_c - h_a)\hat{b}}{h_a} \quad (3.28g)$$

$$\Leftrightarrow \frac{h_a(e^{\nabla h} - 1)}{h_c - h_a} = \hat{b} \quad (3.28h)$$

Comme  $\ell_h(\gamma) = \ell_h(ab) + \ell_h(bc) = 2$ , on a de même :

$$\hat{c} = \frac{h_a(e^{2\nabla h} - 1)}{h_c - h_a} \quad (3.29)$$

En combinant (3.28) et (3.29), le ratio de taille s'obtient comme suit :

$$\frac{\|bc\|}{\|ab\|} = \frac{\hat{c} - \hat{b}}{\hat{b} - \hat{a}} \quad (3.30a)$$

$$= \frac{(e^{2\nabla h} - 1) - (e^{\nabla h} - 1)}{(e^{\nabla h} - 1) - (e^0 - 1)} \quad (3.30b)$$

$$= \frac{e^{2\nabla h} - e^{\nabla h}}{e^{\nabla h} - 1} \quad (3.30c)$$

$$= \frac{e^{\nabla h}(e^{\nabla h} - 1)}{e^{\nabla h} - 1} \quad (3.30d)$$

$$\boxed{s(b) = \frac{\|bc\|}{\|ab\|} = e^{\nabla h}} \quad (3.30e)$$

En bornant le ratio de taille d'un point quelconque  $p$  par  $\epsilon_1$ , on a :

$$s(p) = \epsilon_1 \quad (3.31a)$$

$$\Leftrightarrow e^{\nabla h} = \epsilon_1 \quad (3.31b)$$

$$\Leftrightarrow \nabla h = \ln |\epsilon_1| \quad (3.31c)$$

$$\Leftrightarrow \frac{h_q - h_p}{\|pq\|} = \ln |\epsilon_1|, \forall q \in \partial S \quad (3.31d)$$

$$\Leftrightarrow \frac{h_q}{h_p} = 1 + \ln |\epsilon_1| \ell_p(pq) \quad (3.31e)$$

$$\Leftrightarrow 1 + \epsilon_2 \ell_p(pq) = 1 + \ln |\epsilon_1| \ell_p(pq) \quad \text{d'après l'équation (3.27)} \quad (3.31f)$$

$$\Leftrightarrow \boxed{\text{donc } \epsilon_2 = \ln |\epsilon_1|} \quad (3.31g)$$

Ainsi borner la variation de  $h$  par  $\epsilon_2$  revient à borner le ratio de taille par  $\ln |\epsilon_1|$ . Enfin, il reste à exprimer  $\epsilon_3$  en fonction de  $\epsilon_2$  et  $\epsilon_1$ . On a :

$$\epsilon_3 = c(pq) \quad (3.32a)$$

$$\Leftrightarrow \epsilon_3 = \left[ \frac{h_q}{h_p} \right]^{1/\ell_h(pq)} \quad (3.32b)$$

$$\Leftrightarrow \epsilon_3 = \exp \left[ \ln \left| \frac{h_q}{h_p} \right| \frac{1}{\ell_h(pq)} \right] \quad (3.32c)$$

$$\Leftrightarrow \epsilon_3 = \exp \left[ \ln \left| \frac{h_q}{h_p} \right| \frac{\nabla h}{\ln \left| \frac{h_q - h_p}{h_p} + 1 \right|} \right] \quad \text{d'après l'équation (3.28f)} \quad (3.32d)$$

$$\Leftrightarrow \epsilon_3 = \exp \left[ \ln \left| \frac{h_q}{h_p} \right| \frac{\nabla h}{\ln \left| \frac{h_q}{h_p} \right|} \right] \quad (3.32e)$$

$$\Leftrightarrow \epsilon_3 = e^{\nabla h} \quad (3.32f)$$

$$\Leftrightarrow \ln |\epsilon_3| = \nabla h \quad (3.32g)$$

$$\Leftrightarrow \boxed{\text{donc } |\epsilon_3| = |\epsilon_1|} \quad \text{d'après l'équation (3.31c)} \quad (3.32h)$$

En combinant les équations (3.31) et (3.32), on a bien :

$$\epsilon_2 = \ln |\epsilon_1| = \ln |\epsilon_3| \quad (3.33)$$

Par conséquent, le choix d'une mesure de gradation parmi celles décrites en (3.26) est arbitraire, puisque borner l'une d'entre elles permet de borner les deux autres.

NOS CHOIX. Par souci de simplicité, nous décidons de contrôler la  $h$ -variation telle que  $\frac{1}{\epsilon} \leq \nabla h_p \leq \epsilon$  en tout point  $p$  de la triangulation. Pour le lissage des directions, nous optons pour l'approche décrite dans [101] basée sur le recours à l'intersection de tenseurs.

- **ajustement de tailles.** Pour chaque arête  $[pq]$  incidente à  $p$ , la densité  $\rho_i(p)$  associée est lissée par un facteur  $\lambda$  tel que :

$$\lambda = \frac{\rho_i(p)}{\rho_i(q)} = (1 + \epsilon_2 \ell_p(pq))^{-2} \quad (3.34a)$$

$$= (1 + \epsilon_2 h_p(pq) \|pq\|)^{-2} \quad (3.34b)$$

Ainsi dans le cas anisotrope, il nous suffit d'appliquer un facteur de lissage  $\lambda_i$  pour chaque direction principale  $\mathbf{v}_i$ . Ainsi le tenseur métrique  $g_p$  est mise à jour comme suit :

$$\forall \mathbf{u}, \mathbf{v} \in T_p \Gamma, g_{p|\mathbf{w}}(\mathbf{u}, \mathbf{v}) = \langle \mathbf{u}, J_p \mathbf{P}^T \eta_{p|\mathbf{w}} \mathbf{D} \mathbf{P} J_p^T \mathbf{v} \rangle \quad (3.35a)$$

$$\text{avec } \eta_{p|\mathbf{w}} = \begin{pmatrix} (1 + \epsilon_2 h_{1,p} \|\mathbf{w}\|)^{-2} & 0 \\ 0 & (1 + \epsilon_2 h_{2,p} \|\mathbf{w}\|)^{-2} \end{pmatrix} \quad (3.35b)$$

- **ajustement de directions.** Elle s'effectue en deux phases :

- EXPANSION : chaque point  $p$  propage sa métrique  $g_{q|\mathbf{w}}$  en tout point  $q$  de la surface tel que  $\mathbf{w} = \vec{pq}$ , selon un facteur relatif aux coefficients  $\lambda_{i,\mathbf{w}}$  décrit en (3.35). Ainsi plus on s'éloigne de  $p$ , plus le facteur de lissage est atténué (puisque  $\|\mathbf{w}\|$  croît), et donc plus l'influence de  $p$  décroît.
- RÉDUCTION : la métrique finale  $g_p$  au point  $p$  s'obtient par intersection de toutes les métriques  $g_{p|\mathbf{w}}$  qu'il a reçues de chaque point  $q$  de la variété selon les formules décrites en (3.35) et (A.18) (page 139). Notons que chaque  $h_{i,p}|\mathbf{w}$  correspond à la longueur de l'axe  $\mathbf{v}_i$  de la boule associée à  $g_{p|\mathbf{w}}$  comme sur la figure 3.21. En prenant le plus grand ellipsoïde strictement contenu dans l'intersection de ces boules, on ne garde que la contrainte de taille la plus restrictive  $h_{i,p} \leq \min_{\mathbf{w}=\vec{pq}} h_{i,p}|\mathbf{w}$  pour chaque direction  $\mathbf{v}_i$  de chaque métrique  $g_{p|\mathbf{w}}$ .

En notant  $n$  le nombre de points, le tenseur métrique final associé au point  $p$  s'obtient par :

$$\forall \mathbf{u}, \mathbf{v} \in T_p \Gamma, g_p(\mathbf{u}, \mathbf{v}) = \langle \mathbf{u}, J_p ((\bigcap_{i=1}^n g_{p|\mathbf{w}_i}) \cap g_p) J_p^T \mathbf{v} \rangle. \quad (3.36)$$

EN PRATIQUE. Le problème avec cette procédure est qu'elle est en  $O(n^2)$ ,  $n$  étant la résolution de la triangulation. En pratique, c'est réellement problématique car  $n$  est très grand. Pour le contourner, la réduction n'est appliquée que sur le voisinage du point courant  $p$  par le biais des arêtes  $\mathbf{w}$  incidentes à  $p$ . Ainsi les phases d'expansion et de réduction sont faites simultanément, puisque chaque  $p$  notifie son voisin  $q$  si  $g_p$  a été mis à jour, et vice-versa. Enfin, les deux phases sont propagées jusqu'à convergence, i.e jusqu'à ce que le champ de métriques ne soit plus modifié, ou si un nombre max d'itérés est atteint. La procédure complète est décrite à l'algorithme 3.8.

### 3.3.2 Transport de métriques

CADRE ET BUT. À ce stade, nous avons construit un champ tensoriel qui encode la répartition idéale des points et l'alignement des mailles afin de minimiser l'erreur de la surface ou celle d'une solution numérique  $\mathbf{u}$  calculée sur la triangulation, conformément à la résolution souhaitée et à la sensibilité de la norme utilisée. De plus, il nous garantit une variation lisse des tailles de toute paire d'arêtes incidentes. Lorsqu'un point est créé ou déplacé, sa normale ainsi que son tenseur métrique doivent être recalculés ou interpolés à partir de ses voisins, puisqu'ils ont changés. Néanmoins l'interpolation répétée d'un tenseur métrique implique une perte d'anisotropie à cause des effets de diffusion<sup>71</sup>. Par

<sup>71</sup>Intuitivement, quand on prend la moyenne d'une moyenne d'une moyenne alors on perd forcément en précision. C'est au moins le cas quand on effectue une interpolation linéaire de tenseurs métriques.

Algorithme 3.8: Gradation

```

 $t = 0$ , et marquer tous les points comme étant à traiter.       $\triangleright$  grâce à une file de tâches.

répéter
    pour chaque point  $p$  marqué faire
        calculer une base locale  $J = (\partial_u p, \partial_v p)$  du plan tangent de  $p$ .
        calculer  $M_p$  le tenseur associé à  $g_p$  dans la base  $J$ .
        diagonaliser  $M_p$  et extraire  $h_{0,p}$  et  $h_{1,p}$ .
        pour chaque point voisin  $q$  de  $p$  faire
            Expansion de la métrique de  $q$  vers  $p$  par le biais de  $g_{p|\mathbf{w}}$ 
             $N = R = P = D = \lambda = 0_{2 \times 2}$ 
            calculer  $\tilde{g}_q$  par transport parallèle de  $g_q$  vers  $p$ .
            calculer  $M_q$  le tenseur associé à  $\tilde{g}_q$  dans la base  $J$ .
            calculer  $N = M_p^{-1} M_q$  et diagonaliser  $N = R^{-1} \Lambda R$ .       $\triangleright$  voir A.2.3
            pour chaque direction  $\mathbf{v}_i \in R$  faire
                 $\eta_i = (1 + \epsilon h_{i,p} \|\mathbf{w}\|)^{-2}$ .       $\triangleright$  voir équation 3.27
                 $\lambda_{1,i} = \langle \mathbf{v}_i, M_p \mathbf{v}_i \rangle$ .       $\triangleright$  voir équation A.16
                 $\lambda_{2,i} = \langle \mathbf{v}_i, \eta_i M_q \mathbf{v}_i \rangle$ .       $\triangleright$  voir équation 3.35
            fin
            Réduction avec la métrique reçue  $g_{p|\mathbf{w}}$ 
            si il y a un  $j$  pour lequel  $\lambda_{1,j} < \lambda_{2,j}$  alors
                pour chaque direction  $\mathbf{v}_i$  faire
                     $D_{ii} = \min(\max(\lambda_{1,i}, \lambda_{2,i}), h_{\max}^{-2})$ .       $\triangleright$  voir équation A.17
                     $P = R^{-1}$ 
                    stocker  $g_p = J P^T D P J^T$ .       $\triangleright g_{p|\mathbf{w}} \cap g_p$ , voir équation A.18
                fin si
                marquer  $q$ .
            fin
            si  $g_p$  a été modifié, démarquer  $p$ .
        fin
         $t = t + 1$ 
jusqu'à aucun point marqué ou que  $t \geq t_{\max}$ 

```

ailleurs, déplacer un tenseur métrique sur la surface peut dévier ses directions principales. Pour minimiser cette déviation, nous utilisons une routine de *transport parallèle*<sup>72</sup> au sens de la définition 29 mais étendu au cas des tenseurs métriques. Intuitivement, il généralise la notion de *translation* de vecteurs au cas des variétés. Notons que cela est déjà utilisé dans MMGS de manière heuristique. Ici, notre but est de prouver formellement qu'un transport de tenseurs métriques préservant les directions peut être réalisé par un simple transport parallèle de vecteurs tangents (propriété 3).

**Définition 29** (TRANSPORT PARALLÈLE). Soit  $\Gamma$  une variété munie d'une connexion affine  $\nabla^a$ . Un champ vectoriel  $\mathbf{v}^{[t]}$  du fibré tangent<sup>b</sup> de  $\Gamma$  le long d'une courbe  $\gamma : I \rightarrow \Gamma$  est dit parallèle relativement à cette connexion si  $\nabla_{\dot{\gamma}[t]} \mathbf{v}^{[t]}$  s'annule pour tout  $t \in I$ .

Ainsi un vecteur tangent  $\mathbf{v}$  est transporté parallèlement de  $p$  à  $q$  sur  $\gamma$  si le champ vectoriel  $\mathbf{v}^{[t]}$  induit par son déplacement le long de  $\gamma$  est parallèle.

<sup>a</sup>Une *connexion affine* sur une variété différentielle  $\Gamma$  définit la notion de dérivée d'un champ vectoriel  $\mathcal{U}$  par rapport à un autre champ vectoriel  $\mathcal{V}$  du fibré tangent de  $\Gamma$ . Son développement décrit une courbe  $\gamma$  reliant les plans tangents de deux points de  $\Gamma$ . Dans le cadre du transport parallèle,  $\mathcal{V}$  correspond au champ de vitesse de  $\gamma$ .

<sup>b</sup>Le fibré tangent de la variété  $\Gamma$  est l'union disjointe de tous les plans tangents en tout point de  $\Gamma$ .

<sup>72</sup>Notons juste qu'elle fait intervenir des notions rigoureuses de géométrie différentielle assez complexes à appréhender : nous veillerons à rester synthétique en ne détaillant que les points pertinents dans notre contexte.

**Définition 30** (CONNEXION DE LEVI-CIVITA). *Il existe une unique connexion affine  $\nabla$ , dite de LEVI-CIVITA, sur une variété riemannienne  $(\Gamma, g)$  telle que :*

- elle soit sans-torsion<sup>a</sup>, i.e que la rotation des plans tangents autour d'une géodésique  $\gamma$  est nulle quand ils sont transportés parallèlement le long de  $\gamma$  par le biais de  $\nabla$ .
- elle soit compatible<sup>b</sup> avec  $g$ , i.e que le produit scalaire local en tout point  $p$  de  $\Gamma$  est préservé par transport parallèle de  $p$  le long de  $\gamma$  : ce déplacement est donc une isométrie.

*Si la variété est munie de deux métriques  $g_1$  et  $g_2$  alors leurs connexions de LEVI-CIVITA ne sont pas nécessairement les mêmes<sup>c</sup>.*

<sup>a</sup>Elle est sans torsion si pour tout champs vectoriels  $\mathcal{U}, \mathcal{V}$  du fibré tangent de  $\Gamma$ , on a :  $\nabla_{\mathcal{U}}\mathcal{V} - \nabla_{\mathcal{V}}\mathcal{U} = [\mathcal{U}, \mathcal{V}]$ , où  $[\cdot, \cdot]$  désigne le crochet de Lie. Le terme de gauche quantifie le manque de commutativité du transport parallèle induite par  $\nabla$ , et le terme de droite le manque de commutativité des flots associés à  $\mathcal{U}$  et  $\mathcal{V}$ , qui est intrinsèque à la variété. Ainsi  $\nabla$  ne "rajoute" pas de torsion supplémentaire que celle intrinsèque à la variété.

<sup>b</sup>Cela implique que :  $(\nabla_{\mathcal{U}}g)(\mathcal{V}, \mathcal{W}) = \partial_{\mathcal{U}}g(\mathcal{V}, \mathcal{W}) - g(\nabla_{\mathcal{U}}\mathcal{V}, \mathcal{W}) - g(\mathcal{V}, \nabla_{\mathcal{U}}\mathcal{W}) = 0$  pour tout  $\mathcal{U}, \mathcal{V}, \mathcal{W}$  du fibré tangent de  $\Gamma$ . On dit alors que  $\nabla$  est une connexion métrique avec  $\nabla g = 0$ .

<sup>c</sup>Car on a deux variétés riemanniennes distinctes dans ce cas.

CAS VECTORIEL. En fait, un vecteur tangent  $\mathbf{v}$  peut être transporté parallèlement d'un point de la variété munie d'une connexion  $\nabla$ , à un autre si le chemin qu'elle emprunte est une géodésique de cette connexion, c'est-à-dire une courbe la variété correspondant au plus court chemin entre ces deux points selon  $\nabla$ . À vrai dire, la déviation de  $\mathbf{v}$  va dépendre de la connexion que l'on choisit. Intuitivement, la connexion de LEVI-CIVITA (voir définition 30) fournit une manière optimale de réaliser le transport de  $\mathbf{v}$  puisqu'elle sans torsion et préserve le produit scalaire local. Mais le plus important est que les géodésiques d'une connexion de LEVI-CIVITA coïncident avec celles de la surface elle-même. Ainsi, le problème revient concrètement à :

- trouver une géodésique "naturelle"  $\gamma$  passant par les sommets de toute arête  $[pq]$ ,
- déterminer une manière de translater un vecteur tangent de  $p$  vers  $q$  le long de  $\gamma$ .

Ici,  $\gamma$  peut être approchée par une B-spline (voir section 3.2.3, page 61). Par contre, déplacer parallèlement  $\mathbf{v}$  le long de  $\gamma$  revient à résoudre l'équation différentielle suivante :

$$\text{trouver le champ vectoriel } \mathbf{v}^{[t]} \text{ du fibré tangent de } \Gamma \text{ tq : } \nabla_{\dot{\gamma}[t]} \mathbf{v}^{[t]} = 0, \text{ pour tout } t \in I. \quad (3.37)$$

Au lieu de résoudre (3.37), elle peut être approchée par le biais de l'échelle de SCHILD [160]. Pour déplacer le vecteur  $\mathbf{v}$  d'un point  $p$  à un point  $q$  le long de la courbe  $\gamma$ , l'idée est de construire une série de parallélogrammes courbes tel qu'un côté de chaque parallélogramme représente la discréttisation de  $\gamma$ , et le côté adjacent représente le déplacement de  $\mathbf{v}$  sur chaque point contrôle de cette discréttisation. De manière concrète, cela se fait en quatre étapes :

- calculer la courbe  $\gamma_1$  de début  $p$  et de vitesse initiale  $\mathbf{v}$  :  $\gamma_1(0) = p, \dot{\gamma}_1(0) = \mathbf{v}, \ell(\gamma) = 1$ .
- calculer la courbe  $\gamma_2$  de début  $r = \gamma_1(1)$  et de fin  $q$ , ainsi que son point milieu  $m$ .
- calculer la courbe  $\gamma_3$  de début  $p$  passant par  $m$  telle que  $m$  soit le milieu de  $\gamma_3$ .
- calculer la courbe  $\gamma_4$  de début  $q$  et de fin  $\gamma_3(1)$ . En fait,  $\dot{\gamma}_4(0)$  est le transporté de  $\mathbf{v}$  sur  $q$ .

En fait, cette construction n'est valide que si  $p$  et  $q$  sont suffisamment proches. Si ce n'est pas le cas, il suffit d'itérer la procédure et donc de construire une "échelle".

CAS TENSORIEL. Ne perdons pas de vue que notre but réel est de réaliser le transport parallèle d'un tenseur métrique  $g_p$  d'un point  $p$  vers un point  $q$  de la variété. De la même manière que pour un vecteur tangent, transporter parallèlement  $g_p$  sur une courbe  $\gamma : I \rightarrow \Gamma$  sous-tendue par  $[pq]$  consiste à trouver un champ tensoriel  $g_{\gamma[t]}$  tel que  $\nabla_{\dot{\gamma}[t]} g_{\gamma[t]} = 0$  pour tout  $t \in I$ , et donc de résoudre l'équation différentielle associée.

Au lieu de résoudre directement ce problème, on va plutôt le ramener à un transport parallèle de vecteurs tangents. Ainsi, cela permet de réutiliser l'échelle de SCHILD pour le transport de  $g_p$ . Notre souci est qu'il faut construire la connexion associée à la variété riemannienne  $(\Gamma, g)$  de sorte qu'elle soit compatible avec  $g$ . Au lieu de l'expliciter, on va plutôt essayer de l'exprimer avec celle de LEVI-CIVITA. L'avantage est qu'elle préserve le tenseur métrique induit de  $\mathbb{R}^3$  lors du déplacement le

long d'une géodésique  $\gamma$ . En fait, si  $P$  est une base orthonormale de  $T_p\Gamma$ , alors il suffit de transporter parallèlement chaque vecteur colonne de  $P$  le long de  $\gamma$  pour obtenir une base orthonormale de  $T_q\Gamma$ . Ici on va montrer que c'est également le cas si on considère les directions principales  $R = (v_1, v_2)$  de  $g_p = R^T D R$  (propriété 3).

**Propriété 3** (TRANSPORT DE MÉTRIQUES). *Notons  $(\Gamma, g)$  notre variété riemannienne.*

*Réaliser le transport parallèle d'un tenseur métrique  $g_p$  d'un point  $p$  vers un point  $q$  revient au transport parallèle de ses directions principales le long d'une géodésique "naturelle"<sup>a</sup> de  $\Gamma$ .*

<sup>a</sup>En fait, il s'agit de géodésiques relatives à la connexion de LEVI-CIVITA associée à la métrique intrinsèque de la surface (ou première forme fondamentale). Cette dernière définit le produit scalaire usuel sur la surface.

PREUVE. Pour démontrer la propriété 3, il faudrait d'abord montrer que la connexion de LEVI-CIVITA associée à la métrique intrinsèque de la surface est bien compatible avec notre métrique  $g_p$  en tout point  $p$  de la surface. Pour toute courbe  $\gamma : I \rightarrow \Gamma$ ,  $g_{\gamma[t]}$  est compatible avec  $\nabla$  si pour tout champ vectoriel  $\mathbf{u}, \mathbf{v}$  du fibré tangent de  $\Gamma$ , on a :

$$(\nabla_{\dot{\gamma}[t]} g_{\gamma[t]})(\mathbf{u}, \mathbf{v}) = \partial_{\dot{\gamma}[t]}(g_{\gamma[t]}(\mathbf{u}, \mathbf{v})) - g_{\gamma[t]}(\nabla_{\dot{\gamma}[t]}\mathbf{u}, \mathbf{v}) - g_{\gamma[t]}(\mathbf{u}, \nabla_{\dot{\gamma}[t]}\mathbf{v}) = 0, \quad \forall t \in I \quad (3.38)$$

En fait,  $g_p$  est une forme bilinéaire symétrique correspondant à une matrice symétrique  $M$  exprimée dans une base locale de  $T_p\Gamma$ . En particulier, il existe une base orthonormale  $P$  donnée de  $T_p\Gamma$  telle que  $M$  est congruente à une matrice diagonale  $D$  dans  $P$ . Autrement dit,  $g_p(v_i, v_j) = 0$  pour tout  $v_i, v_j \in P$ . Dans ce cadre, on va montrer que les valeurs propres  $\lambda_i \in D$  sont invariantes par transport parallèle des vecteurs colonnes  $v_i$  de  $P$  le long d'une courbe  $\gamma : I \rightarrow \Gamma$  sous-tendue par  $[pq]$ . Pour un  $t \in I$  donné, notons  $w^{[t]}$  le transporté d'un vecteur tangent  $w$  au point  $\gamma(t)$ ,  $D$  la matrice diagonale formée des  $\lambda_i$  au point  $p = \gamma(0)$ , et  $P^{[t]} = (v_1^{[t]}, v_2^{[t]})$ . Ainsi, on a :

$$\forall \mathbf{u}^{[0]} \in T_{\gamma[0]}\Gamma, \text{ on a : } g_{\gamma[0]}(\mathbf{u}^{[0]}, \mathbf{v}_2^{[0]}) = \langle \mathbf{u}^{[0]}, M_{\gamma[0]} \mathbf{v}_2^{[0]} \rangle, \quad \forall t \in I \quad (3.39a)$$

$$= \langle \mathbf{u}^{[0]}, \sum_{i=1}^2 \lambda_i^{[0]} v_i^{[0],T} v_2^{[0]} \rangle \quad (3.39b)$$

$$= \langle \mathbf{u}^{[0]}, \lambda_1^{[0]} v_1^{[0]} (v_1^{[0],T} v_2^{[0]}) + \lambda_2 (v_2^{[0],T} v_1^{[0]}) v_2^{[0]} \rangle \quad (3.39c)$$

$$= \langle \mathbf{u}^{[0]}, \lambda_2^{[0]} \|v_2^{[0]}\| v_2^{[0]} \rangle \quad (3.39d)$$

$$= \lambda_2^{[0]} \langle \mathbf{u}^{[0]}, v_2^{[0]} \rangle \quad (3.39e)$$

$$\text{de même } g_{\gamma[0]}(\mathbf{u}^{[0]}, v_1^{[0]}) = \lambda_1^{[0]} \langle \mathbf{u}^{[0]}, v_1^{[0]} \rangle \quad (3.39f)$$

$$\text{or } \nabla_{\dot{\gamma}[t]}\mathbf{u} = \nabla_{\dot{\gamma}[t]}\mathbf{v}_i = \vec{0}, \quad \forall t \in I. \quad (3.39g)$$

$$\text{donc } g_{\gamma[0]}(\nabla_{\dot{\gamma}[t]}\mathbf{u}, \mathbf{v}_i) + g_{\gamma[0]}(\mathbf{u}, \nabla_{\dot{\gamma}[t]}\mathbf{v}_i) = 0 \quad (3.39h)$$

$$\text{donc } \nabla_{\dot{\gamma}[t]} g_{\gamma[0]}(\mathbf{u}, \mathbf{v}_i) = \partial_{\dot{\gamma}[t]} g_{\gamma[0]}(\mathbf{u}^{[0]}, v_i^{[0]}) \quad \text{voir équation (3.38).}$$

$$\text{donc } \nabla_{\dot{\gamma}[t]} g_{\gamma[0]}(\mathbf{u}, \mathbf{v}_i) = 0 \Leftrightarrow \frac{g_{\gamma[0]}(\mathbf{u}^{[0]}, v_i^{[0]}) - g_{\gamma[0]}(\mathbf{u}^{[0]}, v_i^{[0]})}{\int_0^t \|\dot{\gamma}(s)\| ds} = 0 \quad (3.39i)$$

$$\Leftrightarrow \frac{\langle \mathbf{u}^{[0]}, v_i^{[0]} \rangle}{\int_0^t \|\dot{\gamma}(s)\| ds} (\lambda_i^{[0]} - \lambda_i^{[0]}) = 0 \quad \text{car } \langle \mathbf{u}, \mathbf{v}_i \rangle \text{ constant sur } \gamma.$$

$$\Leftrightarrow \lambda_i^{[0]} - \lambda_i^{[0]} = 0 \quad (3.39j)$$

$$\Leftrightarrow M_{\gamma[0]} = \sum_{i=1}^2 \lambda_i^{[0]} v_i^{[0],T} v_i^{[0]} \quad (3.39k)$$

$$\Leftrightarrow M_{\gamma[0]} = P^{[0],T} D P^{[0]} \quad (3.39l)$$

$$\text{d'où } \nabla_{\dot{\gamma}[t]} g_{\gamma[0]}(\mathbf{u}, \mathbf{v}) = 0 \Leftrightarrow \begin{cases} \nabla_{\dot{\gamma}[t]}\mathbf{u} = \nabla_{\dot{\gamma}[t]}\mathbf{v} = \vec{0} \\ g_{\gamma[0]}(\mathbf{u}, \mathbf{v}) = \langle \mathbf{u}, P^{[0],T} D P^{[0]} \mathbf{v} \rangle \end{cases} \quad (3.39m)$$

Ainsi d'après (3.39), réaliser le transport parallèle d'un tenseur métrique  $g_p$  le long de  $\gamma$  revient à transporter parallèlement ses vecteurs propres  $\mathbf{v}_i$  par le biais de la connexion de LEVI-CIVITA associée à la métrique intrinsèque de la surface (aussi appelé première forme fondamentale). En fait, comme les directions du tenseur doivent être orthogonales en tout point de  $\gamma$ , il suffit de transporter une seule direction  $\mathbf{v}_1$  puis de retrouver  $\mathbf{v}_2$  de sorte que  $\langle \mathbf{v}_1^{[t]}, \mathbf{v}_2^{[t]} \rangle = 0$  pour tout  $t \in I$ . Elle est réalisée par le biais de l'algorithme 3.9.

Algorithme 3.9: Transport parallèle d'un tenseur métrique.

```

fonction TRANSPORT-PARALLÈLE( $\gamma, g_p$ )            $\triangleright \gamma$  : géodésique sous-tendue par  $[pq]$ .
    poser  $\tilde{p} = p$ .                            $\triangleright g_p$  : tenseur à transporter de  $p$  à  $q$ .
    pour chaque pas  $t \in [0, 1]$  faire            $\triangleright \gamma : [0, 1] \rightarrow \Gamma$ 
        poser  $\tilde{q} = \gamma[t]$ .
        calculer des jacobiniennes  $J_{\tilde{p}}$  et  $J_{\tilde{q}}$  et en déduire les normales  $\mathbf{n}(\tilde{p})$  et  $\mathbf{n}(\tilde{q})$ .
        extraire une base  $\mathbf{P} = (\mathbf{v}_1, \mathbf{v}_2)$  de  $T_{\tilde{p}}\Gamma$  tel que  $g_{\tilde{p}} = J_{\tilde{p}} \mathbf{P}^T D \mathbf{P} J_{\tilde{p}}^T$ .
        déterminer le point  $s$  comme suit :
            - soit  $r$  l'extrémité de  $\mathbf{v}_1$ ,                                $\triangleright$  échelle de SCHILD
            - calculer un segment  $[\tilde{r}\tilde{q}]$  et soit  $m$  son milieu,
            - calculer un segment  $[\tilde{p}s]$  tel que  $\|\tilde{p}\tilde{s}\| = 2\|\tilde{p}\tilde{m}\|$ .
            prendre  $\mathbf{v}_1^{[t]} = \overrightarrow{\tilde{q}s}$ , puis  $\mathbf{v}_2^{[t]} = \mathbf{v}_1^{[t]} \times \mathbf{n}(\tilde{q})$  et  $\mathbf{P}^{[t]} = (\mathbf{v}_1^{[t]}, \mathbf{v}_2^{[t]})$ .
            stocker  $g_{\tilde{p}}^{[t]} = J_{\tilde{q}} \mathbf{P}^{[t], T} D \mathbf{P}^{[t]} J_{\tilde{q}}^T$ .
            mettre à jour  $\tilde{p} = \tilde{q}$  et  $\mathbf{n}(\tilde{p}) = \mathbf{n}(\tilde{q})$ .
    fin
    retourner  $g_p^{[1]}$ .
fin

```

\*\*\*

## 3.4 ÉVALUATION NUMÉRIQUE

### 3.4.1 Cadre, but et mesures

En bref, nous avons proposé quatre noyaux locaux ainsi qu'une stratégie complète pour le rééchantillonnage et la régularisation anisotropes de surfaces triangulées, conformément à l'erreur d'une solution numérique ou de la surface elle-même. Ces noyaux concilient les contraintes de localité induite par le hardware tout en restant aussi efficaces que les noyaux de référence. En effet, aucun d'entre eux n'impliquent un voisinage dynamique de points ou mailles, ni de séquence dynamique d'opérations. Par contre, ils s'appuient sur des routines géométriques avancées à savoir :

- une projection de points basée sur le calcul de géodésiques pour le placement précis des points sur la surface idéale lors de leur création ou déplacement,
- une relocalisation de points mixte diffusion-optimisation permettant d'améliorer la qualité des mailles tout en minimisant la déformation de la surface.
- un transport optimal de tenseurs métriques pour préserver leurs directions principales lors du déplacement d'un point ou durant la gradation.

Maintenant le but est de montrer numériquement l'efficacité de chaque feature proposée. À cette fin, les noyaux ont été implémentés en C++11 sous forme d'une bibliothèque baptisée `trigen`. Elle est actuellement en cours d'intégration au sein de GMDS et sera bientôt en open-source.

PLAN. Pour commencer, nous montrons l'efficacité du raffinement et de la relaxation. En particulier, nous évaluons l'impact de notre traitement particulier<sup>73</sup> des arêtes vives sur l'approximation de la surface lors du raffinement, puis l'impact de la relaxation sur l'irrégularité de la triangulation ainsi que sa convergence. Ensuite, nous évaluons la précision de notre opérateur de projection en termes

<sup>73</sup>Notamment le fait que les "ridges" soient dupliqués et que nous utilisons l'opérateur basé sur le calcul de géodésiques pour la projection des points de STEINER.

de déformation de la surface lors de la simplification et du lissage. Dans notre cas, la distribution de l'erreur est représentée par la **distance de Hausdorff locale** à chaque point. L'efficacité de notre lisseur est ensuite démontré par une comparaison numérique avec deux noyaux de référence, en termes de qualité de mailles et de déformation de la surface. Par la suite, nous évaluons la convergence en erreur et qualité quand les noyaux sont combinés dans une boucle de recherche locale (algorithme 3.6). Notons juste que nous priorisons l'adaptation basée sur les courbures de la surface en contexte isotrope et anisotrope, puisqu'elle donne une vision intuitive de l'erreur<sup>74</sup>. Néanmoins nous montrons qu'une adaptation à une solution numérique est également supportée.

**ERREUR DE LA SURFACE.** Pour quantifier l'amélioration ou la dégradation locale de la surface, il nous faut disposer d'une mesure de son erreur sur chaque point de la surface triangulée. Pour cela, nous considérons la **distance de Hausdorff locale** à chaque point. Intuitivement, elle mesure l'écart maximal entre la surface triangulée et une surface de référence, sur un voisinage local de points<sup>75</sup>. En fait, la manière dont on l'évalue dépend du fait qu'on ait une surface de référence explicite ou non. Ainsi si l'adaptation consiste en une simplification ou une régularisation de la surface (i.e  $n_{\max} \leq n$  où  $n$  est le nombre de points de la triangulation initiale), alors nous utilisons le célèbre outil **Metro** [114] intégré à **MESHLAB**. Sinon nous approchons la distance de HAUSDORFF localement sur chaque maille, puis sur chaque point par le biais des mailles qui lui sont incidentes. Pour cela, nous considérons l'erreur d'une maille comme l'écart maximal entre ses points et les points de contrôle du patch qui lui est associée telle que décrite par 3.40.

$$d_H(K, \Gamma_K) \leq \max_{i=0}^2 d(\mathbf{e}_i, \gamma_i), \text{ avec } \begin{cases} \mathbf{e}_i \text{ la } i\text{-ème arête de } K, \\ \gamma_i \text{ la courbe sous-tendue par } \mathbf{e}_i, \\ d \text{ la distance euclidienne.} \end{cases} \quad (3.40)$$

Ainsi, sauf mention contraire, nous approchons l'erreur au voisinage d'un point par le maximum des erreurs des mailles incidentes, telle que décrite par (3.41).

$$\varepsilon_h[p] = \max_{K \in \eta[p]} d_H(K, \Gamma_K), \text{ avec } \begin{cases} d_H \text{ la distance de HAUSDORFF,} \\ \Gamma_K \text{ le patch quartique associé à } K. \end{cases} \quad (3.41)$$

**DISTRIBUTION.** In fine, quantifier l'efficacité de nos noyaux revient à mesurer l'erreur de la surface ou de la solution numérique et la qualité des mailles, ainsi que leur évolution. En particulier, nous sommes intéressés par la distribution statistique de ces quantités sur la surface. Cela nous permet de voir dans quelles régions l'erreur est la plus importante par exemple. Néanmoins nous avons également besoin d'avoir une mesure globale (i.e un scalaire) pour quantifier la convergence des noyaux. Ainsi,

- pour l'erreur : nous utilisons un estimateur en norme  $L^2$  de  $\varepsilon_h$  sur la surface. Il fournit une moyenne de l'erreur tout en prenant compte de la discrétisation, et en particulier, de l'aire des mailles incidentes. En notant  $S = \Gamma$ , l'erreur de la triangulation s'écrit :

$$\|\varepsilon_h\|_{L^2(S)} = \left[ \int_S \varepsilon_h[p]^2 dp \right]^{1/2} \quad (3.42)$$

- pour la qualité : nous évaluons sa médiane sur la triangulation. Elle a l'avantage d'être simple et intuitive, sans être trop sensible aux bruits statistiques contrairement à une moyenne.

### 3.4.2 Précision des noyaux

Nous commençons par estimer l'efficacité des noyaux de raffinement et de relaxation puisqu'elle est plus simple à évaluer pour ces deux là.

<sup>74</sup>En effet l'augmentation ou la réduction de l'erreur se traduit directement par l'amélioration ou la dégradation de la surface triangulée.

<sup>75</sup>Notons qu'évaluer une distance de Hausdorff de manière précise est particulièrement compliquée et coûteuse, ainsi on utilise souvent des méthodes heuristiques dans la littérature.

- Raffinement.** Il est censé toujours améliorer l'approximation de la surface. Ainsi nous avons vérifié si c'est réellement le cas pour une surface avec coins et courbes saillantes (figure 3.22). En particulier, nous cherchons à évaluer si notre manière de traiter les arêtes vives est efficace en pratique. La surface initiale est donnée en (1). Lors de l'insertion d'un point, celui-ci doit être unique pour toute paire de mailles adjacentes. Ici nous voyons que si on recourt uniquement aux patchs des mailles incidentes à une arête vive, alors cela induit une déformation de la surface (2). Par contre, quand on calcule la B-spline à l'aide des normales de la courbe saillante  $\gamma$  préalablement stockées, alors on obtient une projection correcte de ces points comme en (3).<sup>76</sup>.

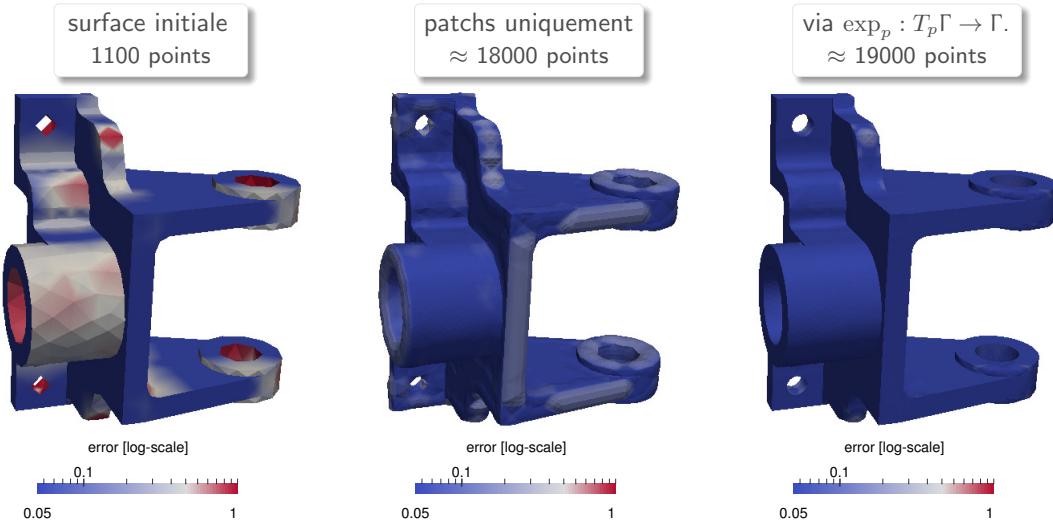


Figure 3.22: Impact de la projection sur l'approximation de la surface lors du raffinement.

- Relaxation des degrés.** Nous avons évalué son impact sur l'irrégularité de la triangulation ainsi que l'évolution des tâches traitées au fur et à mesure des itérés à la figure 3.23. Dans notre cas, l'irrégularité est mesurée comme l'écart absolu entre le degré du point courant et le degré optimal. En fait, la vraie problématique concerne la convergence de ce noyau, voire sa terminaison. En raison des nombreux minima locaux, nous pouvons tomber dans un cas de figure où une arête est basculée indéfiniment. En pratique, nous voyons que le noyau converge et que la distribution est globalement recentrée sur  $d^* = 6$ .

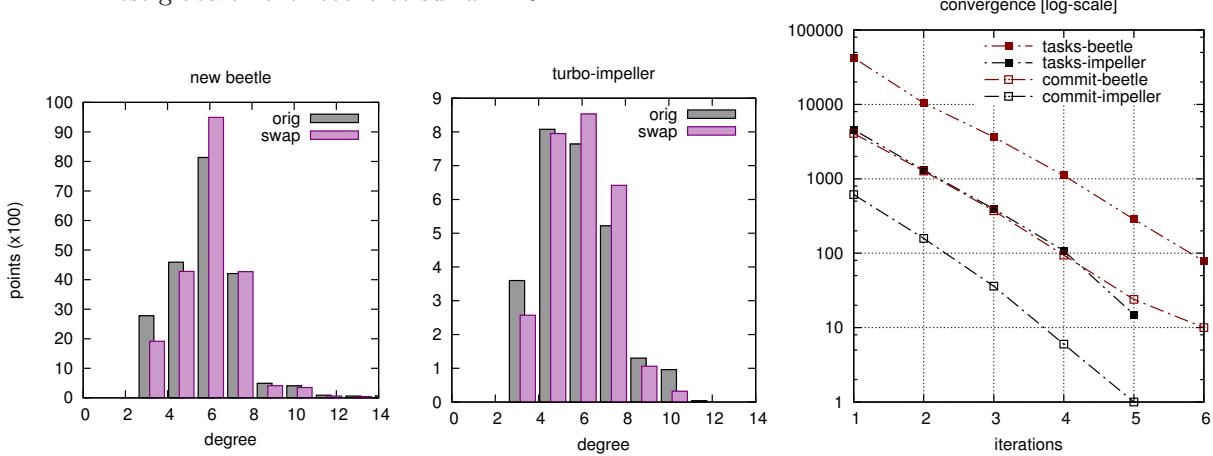


Figure 3.23: Impact de la relaxation sur l'irrégularité de la triangulation.

<sup>76</sup>Notons juste que nous avons souhaité comparer le noyau à celui implémenté dans MMGS qui utilise des PN-triangles pour la reconstruction de la surface. Néanmoins l'écart obtenu n'est pas si significatif, du moins sur les modèles géométriques dont nous disposons. Dans notre cas, le recours aux patchs de GREGORY avec BLENDING des points twists est plutôt motivé par la contrainte de continuité  $G^1$ .

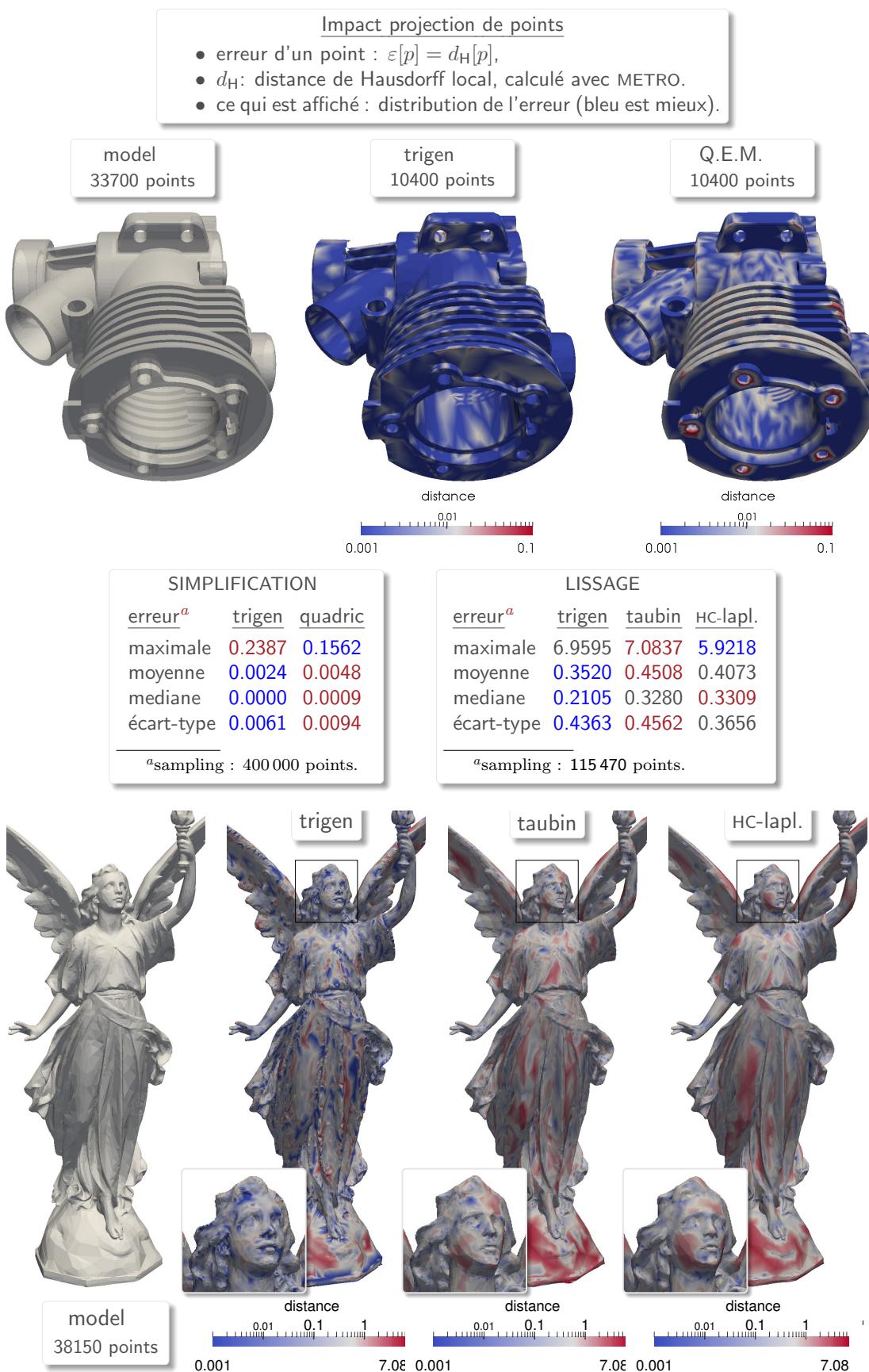


Figure 3.24: Impact de notre opérateur de projection sur la précision des noyaux.

**PROJECTION.** La projection de points basée sur le calcul de géodésiques est notre première contribution majeure (page 64). Pour montrer son efficacité, nous évaluons son impact lors de la simplification et le lissage, en termes d'erreur d'interpolation de la surface (figure 3.24). En fait, le vrai problème de ces deux noyaux réside dans l'inévitable déformation de la surface induite par le déplacement ou la suppression d'un point. Ici, le but est de montrer que nos noyaux basés sur le calcul de géodésiques sont tout aussi précis que les noyaux de l'état de l'art.

- **SIMPLIFICATION.** L'impact de la simplification sur la dégradation de la surface est donnée, avec une comparaison de notre noyau avec le célèbre Quadric Error Metric [115]. Notons que les deux noyaux sont quasi-identiques d'un point de vue algorithmique, la vraie différence réside dans la manière de placer le point résultant sur la surface idéale après la contraction d'une arête. Ici, la déviation maximale des normales est  $\theta_{\max} = 10^\circ$ , et aucune bascule d'arête n'est effectuée. La distance de HAUSDORFF locale est évaluée par le biais du plugin Metro [114] intégré à MESHLAB, avec à peu près 400 000 points d'échantillonnage. Au vu de la distribution de l'erreur, on voit que celle-ci est mieux réduite et équi-répartie dans les régions à forte courbure et sur les courbes saillantes avec notre noyau, contrairement à QEM qui tend à lisser intégralement la surface.
- **LISSAGE.** La déformation induite par notre lisseur comparé à celui de Taubin [116] et du hc-laplacien [117] (via MESHLAB) est donnée dans le cas d'une variété lisse non triviale (figure 3.24). Notons que ces deux dernières méthodes sont optimisées pour minimiser l'effet de rétrécissement induit par un laplacien classique, et contribuent ainsi à réduire la déformation de la surface<sup>77</sup>. Le diamètre autorisé est de  $h_{\max} = 10\%$  de la boîte englobante, la déviation maximale des normales est de  $\theta_{\max} = 7^\circ$ , et près de 115 000 points d'échantillonnage sont utilisés pour l'estimation des distances de HAUSDORFF locales. Ici nous voyons que l'erreur de la surface est mieux réduite à la fois globalement et localement avec notre noyau qu'avec les deux autres.

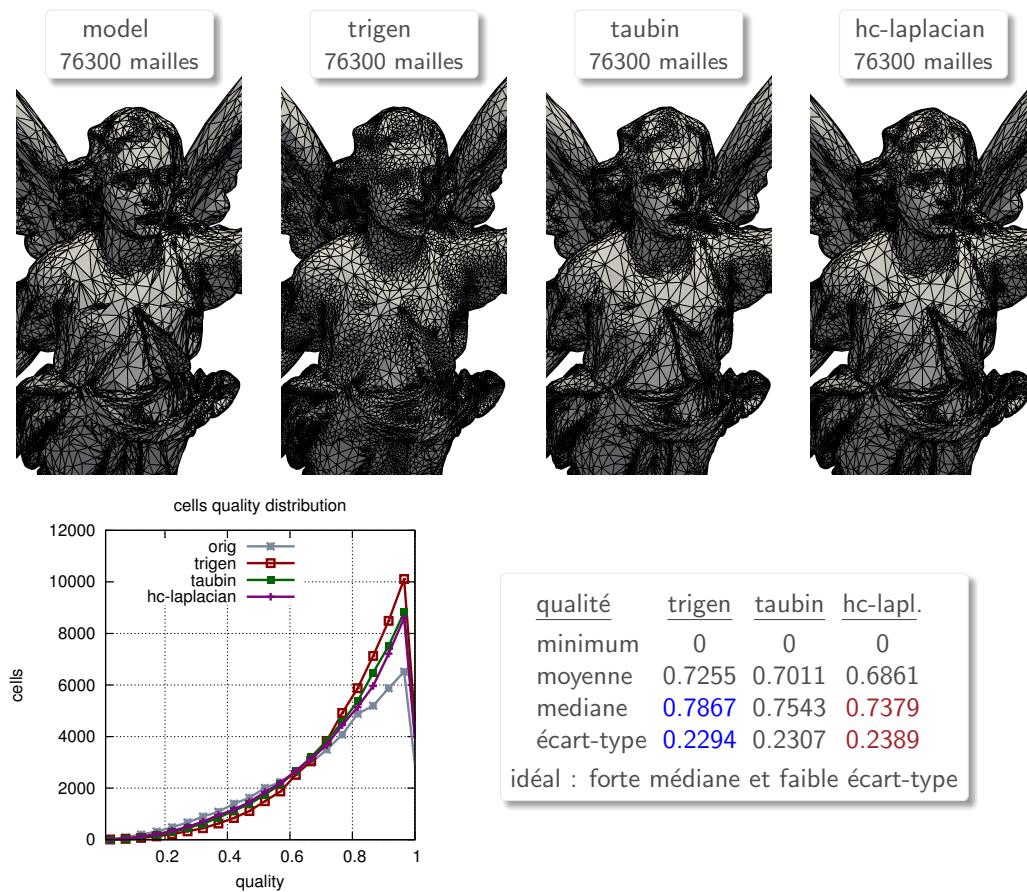


Figure 3.25: Évaluation de notre lisseur en termes de qualité.

<sup>77</sup>Pour le noyau de Taubin, nous avons gardé les paramètres par défaut d'expansion-réduction  $(\lambda, \mu) = (0.5, -0.53)$ .

LISSAGE. Il s'agit de notre seconde contribution majeure (page 70). Rappelons que dans notre cas, le lissage vise uniquement à améliorer la qualité des mailles. Un comparaison de qualité de mailles de notre lisseur avec celui de Taubin et HC-laplacien est donnée à la figure 3.25 : c'est exactement le cas-test de la figure 3.24. Nous avons déjà montré que la déformation de la surface est mieux réduite avec notre lisseur qu'avec les deux autres. Au vu de la distribution de la qualité des mailles, nous avons montré que notre lisseur fournit bien une bonne discréétisation de la surface triangulée tout en préservant au mieux sa déformation.

### 3.4.3 Convergence

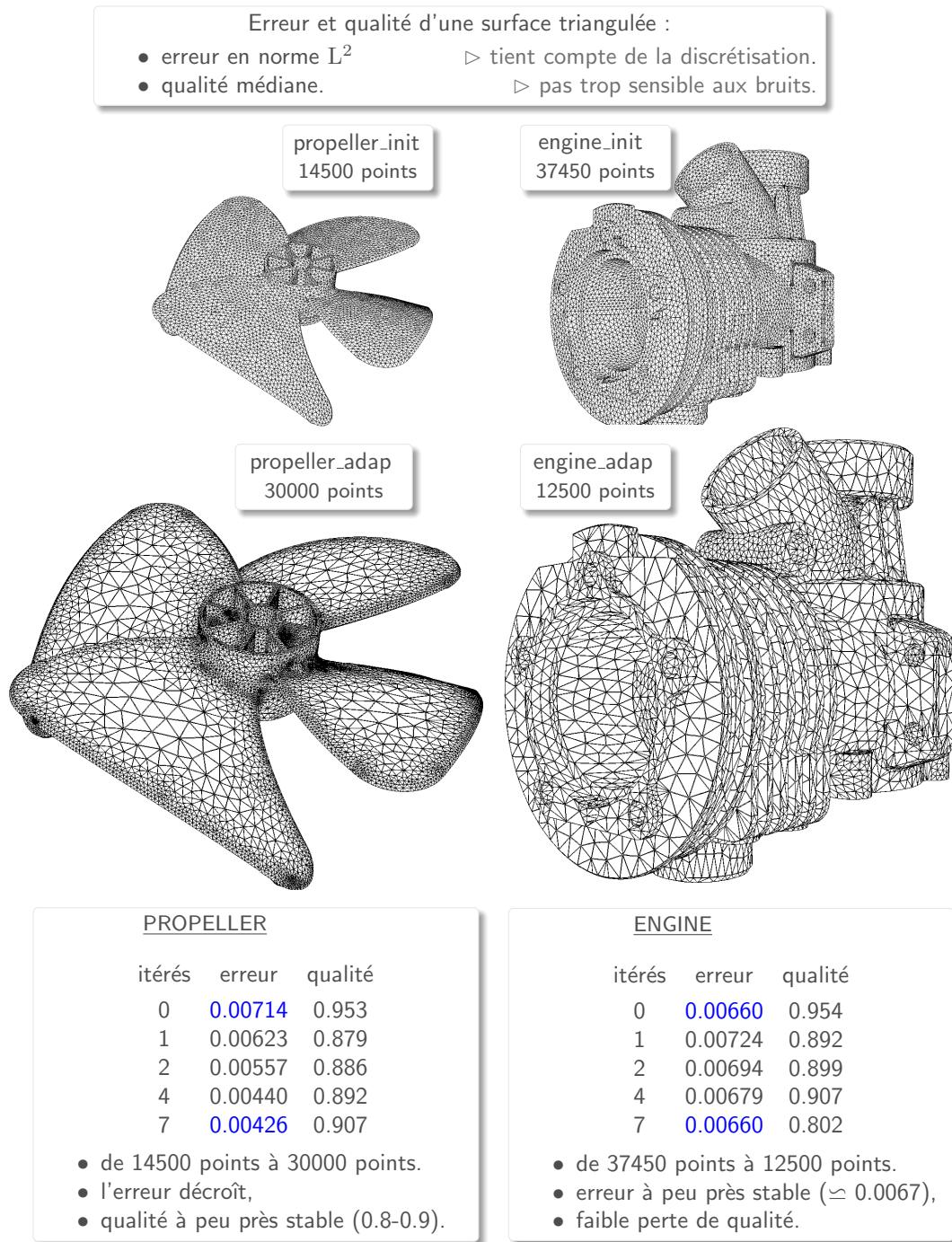


Figure 3.26: Convergence en erreur and qualité quand tous les noyaux sont combinés.

**ALL-IN-THE-BOX.** Cette fois, nous visons à montrer la convergence en erreur et qualité de maillage quand tous les noyaux sont combinés dans une boucle de recherche locale (algorithme 3.6). Plus précisément, nous cherchons à évaluer l'évolution de l'erreur de la surface ainsi que la qualité des mailles quand on augmente ou réduit le nombre de points cible. Pour cela, nous considérons des adaptations basées sur les courbures locales à la figure 3.26, à la fois dans le cas d'un raffinement isotrope (`propeller`,  $n_{\max} = 2n$ ) et d'une simplification anisotrope (`engine`,  $n_{\max} = \frac{n}{3}$ ) de surfaces avec les quatre noyaux combinés. Ici, les courbures locales sont calculées à l'aide du schéma de MEYER [161] (page 137), et un simple estimateur d'erreur en norme infinie est utilisé pour la construction du champ de tenseurs métriques. Enfin, une routine de gradation est utilisé avec un seuil de H-SHOCK fixé à 1.5.

- **OBSERVATIONS.** Pour `propeller`, nous voyons clairement que les points sont majoritairement répartis sur les courbes caractéristiques et les régions à fortes courbures. Pour `engine`, une dégradation de la surface est nécessairement introduite puisqu'on réduit significativement le nombre de points. Néanmoins les courbes caractéristiques sont bien préservées, et les mailles sont correctement étirées le long des directions de courbures minimales.
- **ÉVOLUTION.** L'erreur de la surface et la qualité des mailles sont donnés à chaque itéré. Pour `propeller`, la qualité est relativement préservée tandis que l'erreur décroît au fur et à mesure. Pour `engine`, l'approximation de la surface est relativement préservée malgré un nombre de points réduit d'un facteur trois.

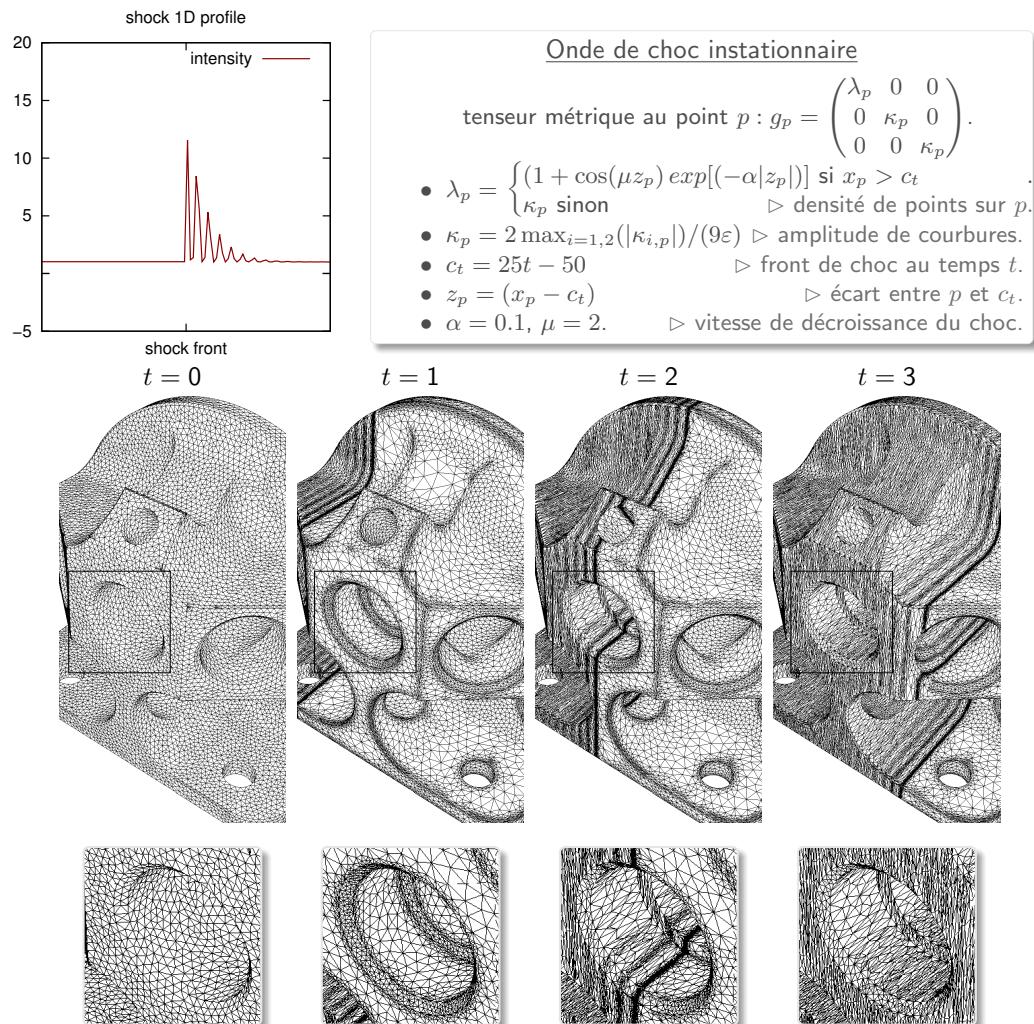


Figure 3.27: Adaptation anisotrope à une densité de points définie par l'utilisateur.

Par ailleurs, le cas d'une adaptation à une métrique définie par l'utilisateur est donnée à la figure 3.27. Ici le but est juste de montrer que toutes les échelles (front de choc et oscillations) sont finement capturées, et qu'on a un étirement correct des mailles qui restent parallèles au front de choc.

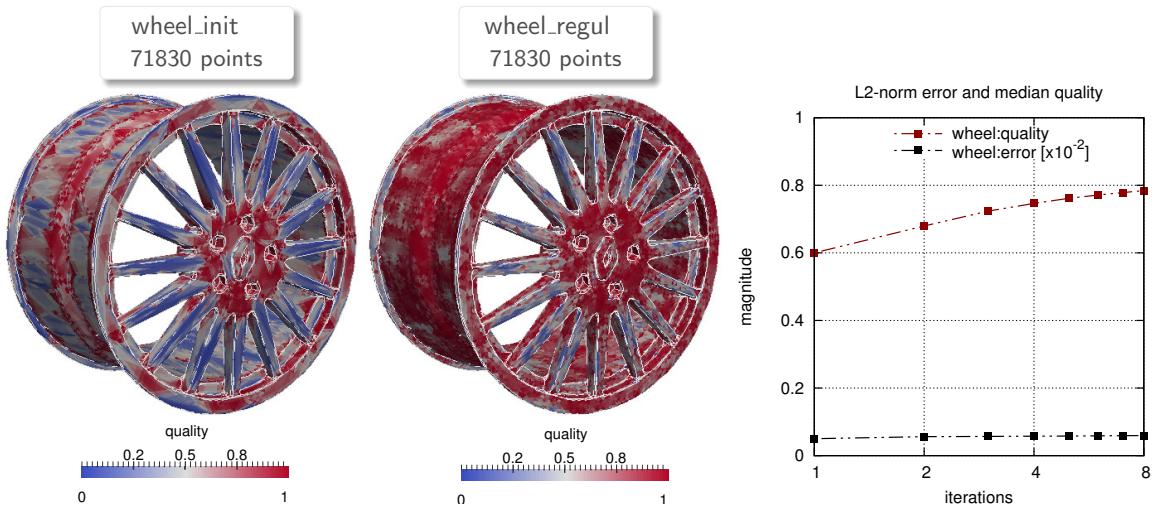


Figure 3.28: Convergence en qualité lors de la phase de régularisation.

**RÉGULARISATION.** Pour finir, nous avons évalué la convergence en qualité des noyaux de régularisation sur une surface dont la discréétisation est fortement bruitée (figure 3.28). Dans notre cas, la boucle de régularisation entrelace les passes de relaxation de degrés et de lissage. Ici, on voit clairement que l'erreur se stabilise tandis que la qualité se voit améliorée au fur et à mesure des itérés.

### 3.5 CONCLUSION

En résumé, nous avons proposé quatre noyaux locaux ainsi qu'une stratégie complète pour l'adaptation de surfaces triangulées à l'erreur d'une solution numérique ou de la surface elle même, à la fois en contexte isotrope et anisotrope. Les noyaux en question respectent les contraintes de localité induites par nos architectures cibles. Rappelons qu'aucun d'entre eux n'implique un voisinage dynamique de points ou de mailles, ni de séquence dynamique et particulière d'opérations (page 66). Pour rester aussi efficaces que les noyaux de référence, ils s'appuient sur :

- une projection de points basée sur le calcul de géodésiques pour le placement précis des points sur la surface idéale lors de leur création ou déplacement (page 64),
- une relocalisation de points mixte diffusion-optimisation qui améliore la qualité des mailles tout en minimisant la déformation de la surface (page 70).
- un transport optimal de tenseurs métriques qui préserve leurs directions principales lors du déplacement d'un point ou durant la gradation (page 87).

L'efficacité de chaque contribution a été soit formellement prouvée soit évaluée numériquement. En effet, la validité du transport optimal de tenseurs métriques a été prouvée (page 90). L'impact de notre méthode de projection de points ainsi que la relocalisation de points ont été mis en évidence par une comparaison avec des noyaux de référence en simplification et lissage [115–117], et cela en termes de déformation de la surface et qualité des mailles (figures 3.24 et 3.25). La convergence et la conformité des noyaux quand ils sont combinés est également montrée à travers des exemples représentatifs impliquant des adaptations basées sur l'approximation de la surface ou une densité définie par l'utilisateur (figures 3.26 à 3.28). Une partie de ces travaux ont été publiés dans un article [RL18] et présentée oralement à un congrès [LR18].

\*\*\*

## Portage des noyaux sur les architectures manycore.

Rappelons que notre but est de fournir un remailleur surfacique qui soit spécifiquement adapté aux machines manycore. À ce stade, nous disposons des noyaux locaux ainsi qu'une stratégie complète pour l'adaptation de surfaces triangulées qui respectent les contraintes de localité induite par le hardware, tout en restant aussi efficaces que l'état de l'art. Ainsi il reste à réaliser leur portage sur nos machines cibles. En raison de leur irrégularité intrinsèque, ces noyaux ne passent pas aisément à l'échelle, surtout sous-contrainte mémoire. Dans ce cadre, nous visons à en fournir une parallélisation efficiente.

### Contributions

- une approche multithread lock-free de bout en bout,
- deux algorithmes pour l'extraction de parallélisme amorphe,
- une approche de structuration des accès-mémoire irréguliers,
- une synchronisation à grain fin pour la consistance de la topologie.

Publications : trois articles [[RLP16](#), [RLP+17](#), [RL18](#)].

4.1	Introduction . . . . .	100
4.1.1	Cadre et contraintes . . . . .	100
4.1.2	Irrégularité des noyaux . . . . .	101
4.1.3	Démarche et contributions . . . . .	103
4.2	Parallélisation des noyaux . . . . .	104
4.2.1	Extraction du parallélisme . . . . .	104
4.2.1.1	Notre approche . . . . .	105
4.2.1.2	Pour la simplification . . . . .	107
4.2.1.3	Pour la relaxation . . . . .	109
4.2.2	Restructuration des accès-mémoire . . . . .	111
4.2.2.1	Refonte en vagues synchrones . . . . .	111
4.2.2.2	Insertions de mailles . . . . .	112
4.2.2.3	Réduction de données . . . . .	113
4.2.3	Consistance des données d'incidence . . . . .	114
4.2.3.1	Contraintes en lock-free . . . . .	115
4.2.3.2	Notre synchronisation . . . . .	116
4.3	Évaluation numérique . . . . .	117
4.3.1	Cadre, architectures et paramètres . . . . .	117
4.3.2	Surcoûts et efficience de nos briques . . . . .	120
4.3.2.1	Extraction de tâches . . . . .	120
4.3.2.2	Notre synchronisation . . . . .	122
4.3.3	Scaling et surcoûts . . . . .	122
4.3.4	Débits de tâches par noyau . . . . .	125
4.4	Conclusion . . . . .	126

## 4.1 INTRODUCTION

### 4.1.1 Cadre et contraintes

BUT. À ce point, nous disposons de noyaux pour l'adaptation anisotrope de surfaces triangulées à l'erreur d'une solution numérique ou de la surface elle-même. Jusqu'ici nos choix de design ont été dictés par la recherche d'une localité maximale<sup>1</sup> pour chaque noyau, tout en restant aussi efficace que l'état de l'art. Le but est maintenant de fournir une parallélisation efficiente de chaque noyau, en dépit de leur irrégularité intrinsèque, et compte-tenu des contraintes spécifiques aux architectures massivement multithread.

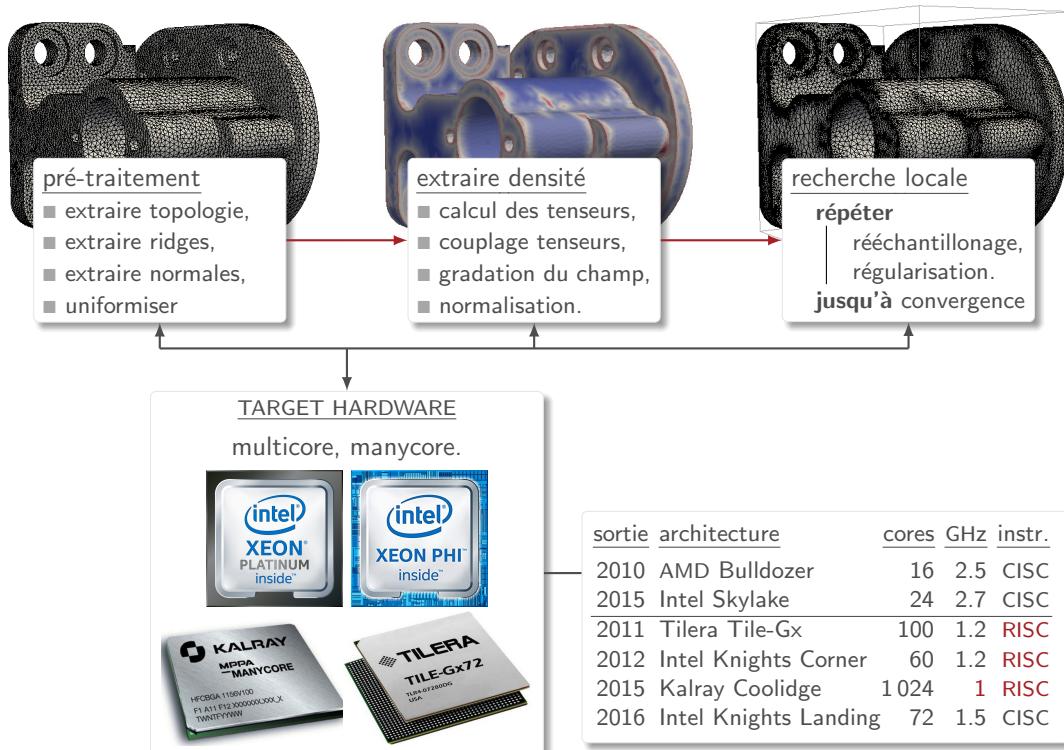


Figure 4.1: Notre objectif.

CONTRAINTEs. En fait le but est de paralléliser chaque phase de la figure 4.1. À ce titre, nous devons tenir compte de deux contraintes liées :

- au **MATÉRIEL** : les machines manycore intègrent un nombre conséquent de cores mais avec un rendement par core faible (GHZ, GFLOPS, GB); tandis que les machines multicore disposent d'une hiérarchie mémoire profonde<sup>2</sup> impliquant une latence d'accès de données pouvant être inégales<sup>3</sup>
- à l'**IRRÉGULARITÉ** : les noyaux de recherche locale sont en réalité des algorithmes **data-driven** et **data-intensive**. Ils sont caractérisés par un parallélisme amorphe, des conflits de tâches non prévisibles, et une faible réutilisation de données en cache. Ainsi ils ne passent pas aisément à l'échelle, surtout sous-contrainte mémoire.

Ici le pré-traitement et l'extraction de la densité ne posent pas de difficulté particulière. Par contre, ce n'est pas le cas des noyaux de recherche locale : on va se concentrer exclusivement sur eux. Pour mieux appréhender le problème, nous commençons par montrer en quoi ces noyaux sont irréguliers et quelles conséquences cela implique (section 4.1.2).

<sup>1</sup>voisinage restreint et statique, pas de séquence dynamique d'opérations.

<sup>2</sup>Ils disposent généralement de trois niveaux de cache, d'une mémoire locale et/ou distante.

<sup>3</sup>Le coût d'un accès de données varie drastiquement selon qu'elle soit en cache, en RAM locale ou distante (page 14).

### 4.1.2 Irrégularité des noyaux

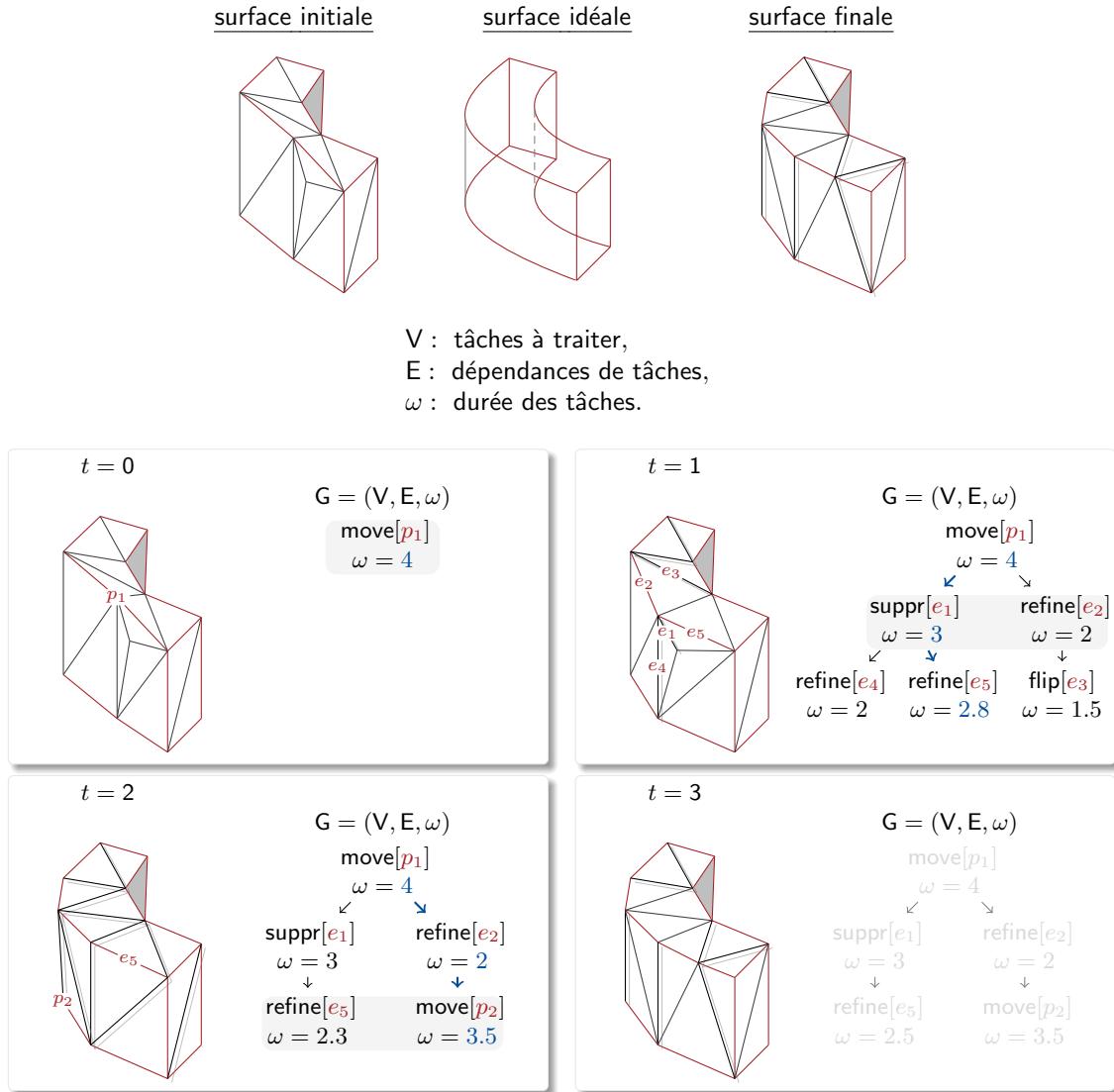


Figure 4.2: Problèmes des dépendances de tâches dynamiques.

DATA-DRIVEN. En fait les tâches relatives aux noyaux, leurs conflits et leurs durées varient au cours de l'exécution. À vrai dire, le fait de traiter une tâche peut générer ou invalider d'autres : deux tâches non conflictuelles à l'itéré courant peuvent ne plus l'être à l'itéré suivant.

Pour se fixer les idées, considérons l'exemple de la figure 4.2 : on part d'une surface initiale et on souhaite obtenir une surface adaptée qui soit plus proche de la surface idéale. Ici, l'évolution de la surface triangulée ainsi que le graphe de tâches est représentée à chaque itéré, et on impose aucune contrainte particulière sur l'entrelacement des noyaux<sup>4</sup>.

- initialement, une seule tâche relative au déplacement de  $p_1$  est prévue;
- une fois traitée, cela a engendré la création de cinq autres tâches à l'itéré suivant, dont deux pouvant être immédiatement traitées (suppr[ $e_1$ ] et refine[ $e_2$ ]);

<sup>4</sup>Notons qu'on aurait le même problème même si on organisait les noyaux par vagues. Ici on a fait le choix d'entrelacer les noyaux pour illustrer rapidement le problème.

- le fait d'avoir traité  $e_1$  et  $e_2$  rend invalide deux tâches prévues  $\text{refine}[e_4]$  et  $\text{flip}[e_3]$  car ces arêtes ont disparues. Par contre, cela a rendu possible le déplacement de  $p_2$ , et la durée de  $\text{refine}[e_5]$  est passée de 2.8 à 2.3 car son voisinage a changé.

Le traitement s'arrête après l'exécution de  $\text{refine}[e_5]$  et  $\text{move}[p_2]$ . Ici, nous avons bien pu remarquer que les tâches disponibles, leurs conflits et leurs durées varient dynamiquement. Ainsi il n'est pas évident de trouver un ordonnancement optimal de tâches qui garantit l'utilisation maximale et équitable des cores.

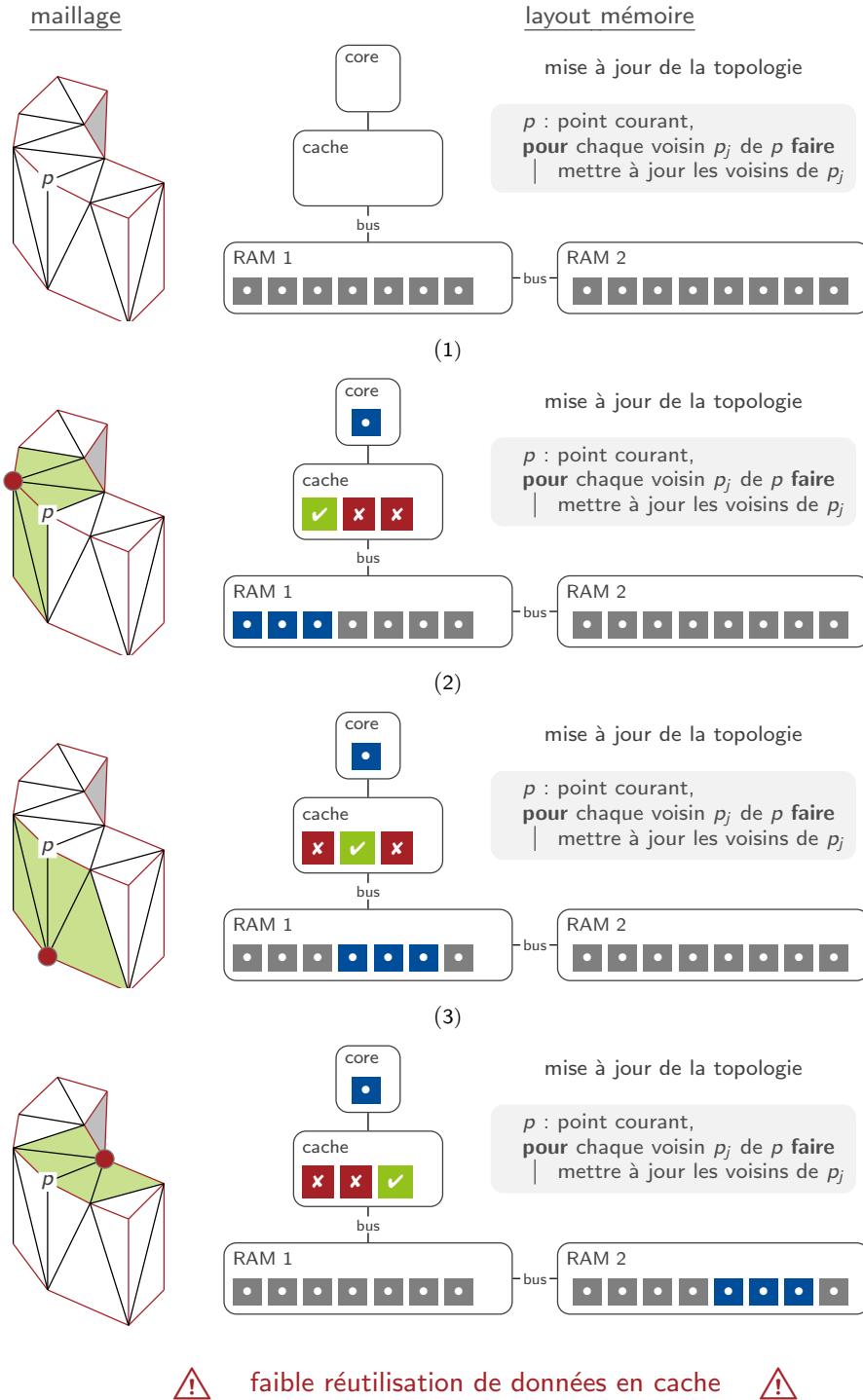


Figure 4.3: Problème des indirections mémoire fréquentes.

**DATA-INTENSIVE.** En fait les instructions des noyaux sont dominés par les accès-mémoire, et qu'on ait une faible réutilisation de données déjà accédées en plus. En d'autres termes, on passe notre temps à accéder aux données, et sur des données différentes<sup>5</sup>. De plus, il n'est pas aisés de préserver un layout de données qui soit *cache-friendly* puisque l'insertion ou de suppression de données ne peuvent pas être inférées.<sup>6</sup>. Pour se fixer les idées, considérons l'exemple de la figure 4.3. Il consiste à la mise à jour des données du voisinage d'un point<sup>7</sup>. Il s'agit d'un cas simplifié avec un seul core, un cache, une RAM locale et une distante.

- initialement, le cache associé au core est vide, et au moment où on traite le premier voisin on va rapatrier un premier bloc de données depuis la RAM locale vers le cache.
- en fait, seule une partie des données rapatriée en cache est utilisée par le core.
- au moment de traiter le second voisin, on est donc obligé de purger le cache puisqu'il est déjà plein, puis de rapatrier un autre bloc de données depuis la RAM. Et là encore, on ne va utiliser qu'une partie des données fraîchement mises en cache.
- enfin, quant au dernier voisin, on va refaire le même traitement sauf que cette fois la donnée rapatriée provient de la RAM distante : les transferts de données vont être encore plus coûteux.

L'exemple considéré est un peu extrême mais c'est assez représentatif des instructions de nos noyaux : en effet, on passe souvent plus de temps à faire des requêtes topologiques qu'à faire du calcul utile. Et pire, on ne réutilise que très peu les données déjà accédées. Ainsi il est impératif de restructurer les accès-mémoire de manière à atténuer la latence induite. Or la difficulté est que les motifs d'accès-mémoire ne sont pas prévisibles car elles dépendent des données<sup>8</sup>.

#### 4.1.3 Démarche et contributions

**TRAVAUX CONNEXES.** En bref, il faudrait exposer des tâches très fines et des accès structurés aux données pour passer à l'échelle sur nos machines cibles. Or les travaux existants en remaillage parallèle implique un parallélisme à gros grain pour la plupart [13]. Ils s'appuient sur un partitionnement de dmaine<sup>9</sup> et une migration de mailles pour le rééquilibrage dynamique de charges. Ainsi l'effort porte essentiellement sur la gestion des interfaces des sous-domaines en vue de réduire le coût des synchronisations, et sur la recherche d'heuristiques pour le rééquilibrage [102]. Des approches à grain fin existent mais impliquent souvent un unique noyau [169], ou traitent les noyaux de manière identique [199] sans tirer profit de leurs spécificités. Dans ce cadre, deux approches ont particulièrement suscité notre intérêt :

- celle de FREITAG [210], qui est la première approche lock-free en adaptation parallèle de maillages. Ici les opérations (raffinement, bascule et lissage) sont exprimées sous forme de graphe, et un stable est extrait pour éviter les conflits, et pour assurer que les données d'incidence restent cohérentes après leur mise à jour. Ici le point critique réside dans l'extraction des tâches puisqu'elle implique un voisinage de distance deux à chaque fois. Un tel choix réduit de manière significative le nombre de tâches pouvant être traitées simultanément<sup>10</sup>.
- celle de ROKOS [199] qui est une extension de celle de FREITAG au cas anisotrope. Il fournit un noyau de simplification en plus, et n'utilise qu'un unique graphe pour tous les noyaux. Pour éviter le recours à un voisinage de distance deux, il utilise un schéma de mise à jour différée des données d'incidence afin de garantir leur consistance. Elle ne tire néanmoins pas profit de la spécificité des noyaux, et ne tient pas compte du fait qu'ils soient data-intensive.

La démarche que nous avons adoptée est une extension de ces deux travaux, mais en tenant compte des spécificités des noyaux et des contraintes hardware.

<sup>5</sup>Ainsi on utilise très peu les données qui sont déjà en cache, et on est obligé de rapatrier les données depuis la RAM : cela induit une latence beaucoup plus élevé et ralentit donc l'algorithme.

<sup>6</sup>En d'autres termes, les patterns d'accès-mémoire ne sont pas prévisibles.

<sup>7</sup>Notons qu'une telle requête est représentative des instructions des noyaux, typiquement la simplification.

<sup>8</sup>Ce qui n'est pas le cas des primitives relatives aux matrices creuses par exemple

<sup>9</sup>en vue de l'exécution des noyaux séquentiels sur chaque sous-domaine

<sup>10</sup>ou degré de parallélisme

**CONTRIBUTIONS.** Nous proposons une parallélisation des noyaux adaptatifs qui concilie leurs aspects **data-driven** et **data-intensive** avec les contraintes induites par le hardware. Elle s'appuie sur trois points :

- **Extraction du parallélisme amorphe.**

Pour gérer les conflits de tâches dynamiques tout en alimentant suffisamment les cores, nous décidons d'extraire le parallélisme amorphe inhérent à chaque noyau à chaque itéré de leur exécution. Pour un noyau donné, nous exprimons les conflits de tâches en un graphe à partir duquel les tâches compatibles seront ensuite extraites. Ici, chaque graphe et l'heuristique associée sont **noyaux-spécifiques** contrairement à l'approche de ROKOS, et qu'on se restreint au **voisinage direct** à tout instant contrairement à celle de FREITAG. En fait, nous proposons deux algorithmes **lock-free** et massivement multithread pour l'extraction de stables et le couplage de graphes non bipartis. Elles font suite à une étude théorique et expérimentale que nous avons mené en apparté.

- **Structuration des accès-mémoire.**

Pour atténuer l'impact des indirections mémoire fréquentes, nous suggérons une manière de structurer les noyaux irréguliers. Pour cela, les tâches inhérentes à un noyau sont organisés en vagues **synchrones**<sup>11</sup>, afin que les écritures en mémoire partagée ne soient pas entrelacés par du calcul local. En fait les insertions et suppression des données topologiques sont coalescées afin de préserver au mieux le placement mémoire initial sans recourir à une rénumérotation d'une part, ainsi que pour atténuer la latence liée aux accès-mémoire distants ou à la saturation fréquente des caches de faible capacité d'autre part. Pour y parvenir, nous proposons deux modes de réduction de données permettant des écritures coalescentes en quasi-asynchrone ou en NUMA-aware.

- **Synchronisation pour les mises à jour de la topologie.**

Pour éviter de recourir à un **voisinage étendu** lors de l'extraction des tâches, nous utilisons un schéma de mise à jour lock-free et différée des données d'incidence comme ROKOS, mais basé sur un recours efficient aux primitives atomiques. En fait le schéma proposé minimise clairement les transferts de données et le surcoût induit comparé à celui de ROKOS.

En fait chaque noyau est implanté par un algorithme non bloquant (**lock-free**) et sans attente (**wait-free**)<sup>12</sup>. Notons que les deux aspects **data-driven** et **data-intensive** des noyaux sont bien pris en compte. Ici les briques algorithmiques proposés sont tous **lock-free**. Enfin, une partie des travaux a été publiée dans trois actes de conférences avec comité de lecture.

**[RLP16]** Fine-grained locality-aware parallel scheme for anisotropic mesh adaptation.

Hoby Rakotoarivelo, Franck Ledoux and Franck Pommereau.

25<sup>th</sup> International Meshing Roundtable, 2016, Washington DC, USA.

**[RLP+17]** Scalable fine-grained metric-based remeshing algorithm for manycore-NUMA architectures.

Hoby Rakotoarivelo, Franck Ledoux, Franck Pommereau and Nicolas Le-Goff.

Euro-Par: 23<sup>rd</sup> International Conference on Parallel and Distributed Computing, 2017, Santiago de Compostella, Spain. [30% of acceptation]

**[RL18]** Accurate manycore-accelerated manifold surface remesh kernels.

Hoby Rakotoarivelo and Franck Ledoux.

27<sup>th</sup> International Meshing Roundtable, 2018, Albuquerque NM, USA.

## 4.2 PARALLÉLISATION DES NOYAUX

### 4.2.1 Extraction du parallélisme

**BUT.** En fait deux types de problèmes doivent être résolus : la **non conformité** de la surface triangulée et l'**inconsistance** de données d'incidence. Pour un noyau donné, les conflits peuvent invalider la

<sup>11</sup>Chaque vague est constituée d'une étape de calcul local, d'écriture de données en mémoire partagée puis d'une barrière de synchronisation [170].

<sup>12</sup>La première propriété assure qu'il y ait au moins un thread qui progresse à tout instant de l'exécution, tandis que la seconde assure que cette exécution se fait en un nombre fini d'itérés.

triangulation : trous induits par un repliement, intersection non conforme d'arêtes par exemple, tandis que les données d'incidence peuvent être corrompues si leur mise à jour n'est pas synchronisée. Sur la figure 4.4, la suppression simultanée de deux points voisins induit un trou dans la surface. Sinon si on contracte deux arêtes voisines vers un même point alors on préserve bien la conformité, néanmoins la liste d'incidence de ce point peut être corrompue si elle est modifiée simultanément.

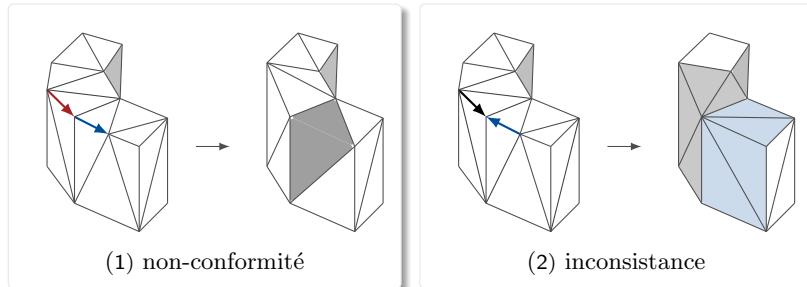


Figure 4.4: Problèmes liés aux data races.

#### 4.2.1.1 Notre approche

In fine le but est d'extraire des tâches compatibles en tenant compte de ces deux contraintes. Il nous faut néanmoins maximiser le nombre de tâches extraites afin d'alimenter suffisamment les cores. Pour concilier ces deux contraintes, nous traitons le problème en deux temps.

- nous considérons uniquement les conflits liés à la **conformité**, afin de maximiser le parallélisme<sup>13</sup>
- nous gérions ensuite la **consistance** des données d'incidence par une synchronisation à grain fin.

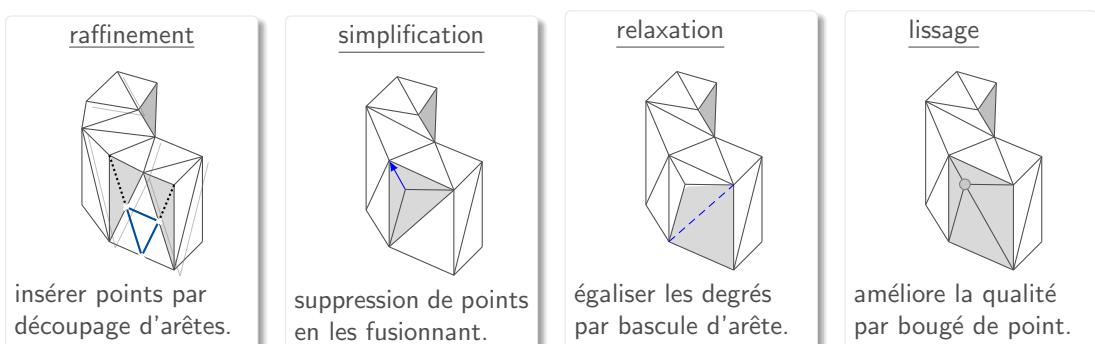


Figure 4.5: Voisinage impacté pour une tâche de chaque noyau.

**CONFLITS.** Pour maximiser le parallélisme, il faudrait une solution noyau-spécifique. En fait les tâches sont inhérentes à la topologie de la triangulation et leurs dépendances évoluent selon comment cette dernière évolue. Ainsi il faut déjà identifier le voisinage impacté pour chacun d'eux (figure 4.5).

- **raffinement** – on a trois motifs de découpage de maille, et quand on coupe une arête d'une maille alors la maille voisine incidente doit également être modifiée.
- **simplification** – si on contracte une arête  $[pq]$  alors les mailles résultantes sont construites par reconnexion des mailles incidentes à  $p$  vers  $q$ , qui ne peuvent être supprimés en même temps.
- **relaxation** – lors de la bascule d'une arête  $[pq]$ , aucune arête incidente à  $p$  ou  $q$  ne peut être basculée en même temps.<sup>14</sup>
- **lissage** – la position du point est soit interpolé à partir de celles ses voisins soit déterminé à partir de la qualité des mailles incidentes : ils ne doivent donc pas être modifiés.

<sup>13</sup>Du coup, nous n'avons pas à considérer un voisinage de distance deux lors de la suppression d'un point par exemple.

<sup>14</sup>Sinon ça induit un croisement d'arêtes et la triangulation n'est plus conforme (définition 4).

APPROCHE. Pour un noyau donné, l'idée est de formuler les tâches par un graphe  $G = (V, E)$ , où  $V$  regroupe les points ou mailles à traiter à l'itéré courant<sup>15</sup>, et  $E$  représente les conflits de points ou les mailles à appairer. Les tâches compatibles sont ensuite identifiées par le biais de trois heuristiques relatives à l'extraction de stable<sup>16</sup>, de couplage<sup>17</sup> maximaux ainsi que la coloration<sup>18</sup> de graphes (notées respectivement `indep`, `match` et `color`, voir figure 4.6).

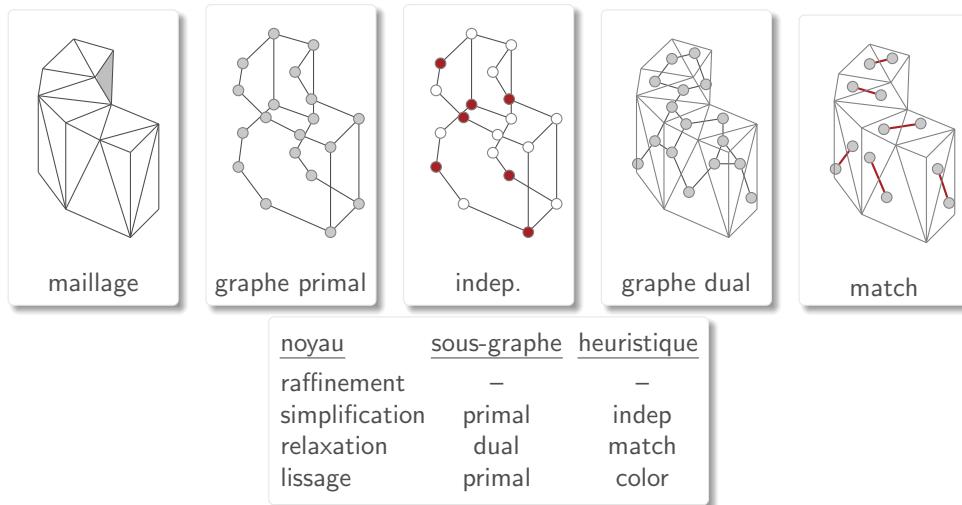


Figure 4.6: Graphes et heuristiques pour l'extraction de tâches.

Bien que la stratégie suggérée s'applique de manière indistincte à tous les noyaux, elle tire profit de leur particularités. Ainsi dans le cas du :

- **raffinement** – le traitement d'une maille implique de figer les mailles voisines, néanmoins ici nous n'utilisons aucun graphe : le noyau est structuré de manière à traiter toutes les mailles actives simultanément.
- **simplification** – comme deux points voisins ne peuvent être supprimés en même temps, ici  $G$  est déduit du graphe primal mais avec les points actifs uniquement : les tâches compatibles sont ensuite extraites à partir d'un stable de  $G$ .
- **relaxation** – le fait de basculer deux arêtes voisines va entraîner un croisement d'arêtes. Pour y remédier, une manière intuitive serait d'extraire un stable d'arêtes. Puisque nous ne disposons pas d'une structure centrée-arête, nous considérons plutôt la paire de mailles incidentes. Ainsi le problème revient à extraire des paires de mailles actives. Ainsi  $G$  est déduit du graphe dual où chaque sommet correspond à une maille, et on extrait ensuite un couplage de  $G$ .
- **lissage** – comme chaque point et ses voisins ne peuvent être déplacés simultanément, ainsi on peut déduire  $G$  du graphe primal à partir des points actifs. Ici la topologie est fixe contrairement au cas de la simplification : ainsi on n'est pas obligé de reconstruire  $G$  à chaque itéré. En fait on peut directement extraire une partition de stables qui va être réutilisée à chaque itéré : cela revient à extraire une coloration de  $G$ .

Pour être applicable, le surcoût lié à ces heuristiques doit être négligeable d'une part, et elles doivent passer correctement à l'échelle d'autre part conformément à la loi d'AMDAHL (page 34). En fait cette phase est cruciale car le degré de parallélisme du noyau en dépend entièrement. Ici nous avons mené une étude comparative expérimentale des heuristiques différentes, afin d'identifier les points critiques compte-tenu de nos contraintes hardware. Ainsi nous proposons deux algorithmes lock-free et massivement multithread pour l'extraction des tâches de simplification et relaxation.

<sup>15</sup>On dit alors que les tâches correspondantes sont **actives**.

<sup>16</sup>Un **stable** d'un graphe est un ensemble de sommets qui soient deux à deux non adjacents.

<sup>17</sup>Un **couplage** d'un graphe est un ensemble d'arêtes de ce graphe qui n'ont pas de sommets en commun.

<sup>18</sup>Une **coloration** d'un graphe est une **partition de stables** de ce graphe.

### 4.2.1.2 Pour la simplification

Pour ce noyau, les tâches actives sont formulées par un sous-graphe  $G = (V, E)$  induit du graphe primal de la triangulation. Ici les points à supprimer sont identifiés par l'extraction d'un stable de  $G$ . En fait ce n'est qu'une étape préliminaire qui est exécuté à chaque itéré de la simplification : il est crucial que son coût reste négligeable. Par ailleurs si on veut extraire un degré de parallélisme élevé afin d'alimenter suffisamment les cores, il faudrait que la qualité de la solution  $\mathcal{U}^*$  reste acceptable.

TRAVAUX CONNEXES. Le calcul de stables est un problème largement étudié notamment en séquentiel. En fournir une version parallèle efficiente reste néanmoins ardue en raison :

- des dépendances intrinsèques et inévitables entre les itérés puisque le stable partiel  $\mathcal{U}^{[t]}$  obtenu à l'itéré  $t$  est construit par complétion ou amélioration de  $\mathcal{U}^{[t-1]}$  [200, 181].
- du fait que les instructions ne consistent qu'à l'accès aux données des sommets alors que le traitement s'effectue de voisinage en voisinage souvent par un parcours en largeur d'abord. Ainsi elles sont dominées par des accès-mémoire, et on a une faible réutilisation des données en cache en raison des multiples indirections<sup>19</sup>[211].

In fine la vraie difficulté dans sa parallélisation réside dans la manière pas trop coûteuse de "casser" les dépendances d'itérés. Une manière d'y parvenir réside dans une sélection probabiliste du prochain voisin à rajouter dans la solution courante  $\mathcal{U}^{[t]}$  : c'est l'approche la plus populaire et utilisée à l'instar de celle de LUBY<sup>20</sup>[182]. Bien que la probabilité que la solution calculée soit incorrecte est non nulle, elle est intéressante car elle converge rapidement<sup>21</sup>. Néanmoins nous ne l'avons pas retenue car :

- elle requiert de régénérer une graine aléatoire par sommet pour converger, ce qui est très coûteux.
- le calcul du sous-graphe induit à chaque itéré peut être coûteux selon l'implémentation<sup>22</sup>.
- sa déterminisation implique le calcul d'un grand nombre premier, qui est réellement coûteux<sup>23</sup>.

En recherche d'une autre alternative, on s'est tourné vers les méthodes spéculatives de coloration de graphes<sup>24</sup>. En fait ce sont des algorithmes synchrones<sup>25</sup> basés sur un schéma d'exécution optimiste : une pseudo-solution est d'abord calculée, et les erreurs sont ensuite résolues dans une étape à part. Elles sont basées sur le noyau séquentiel First-fit qui a l'avantage d'être simple tout en fournissant une solution de qualité acceptable<sup>26</sup>. Ici, le caractère synchrone permet de mieux structurer la gestion des contentions d'accès en mémoire partagée et d'analyser la convergence de ces algorithmes. En fait nous avons mené une étude expérimentale comparative des méthodes existantes [204–206, 200] à la section B.1 en termes de :

- leur sensibilité vis-à-vis de l'irrégularité du graphe en entrée.
- leur scaling et leur sensibilité à la politique d'ordonnancement appliquée.
- leur sensibilité à la localité mémoire et à l'affinité thread-core, notamment en contexte NUMA.
- le ratio de conflits et leur évolution en fonction du nombre de cores.
- la qualité de la solution retournée, et sa dégradation par rapport à la version séquentielle.

En fait l'étude a révélé que les approches spéculatives étaient nettement plus rapides comparées à une approche probabiliste (100 fois en l'occurrence) pour une qualité de solution quasi-identique (de l'ordre de 90% sur RMAT), et quasi-similaire à First-fit.

<sup>19</sup>Enfin on peut avoir un déséquilibre significatif de charges due aux voisnages de tailles inégales.

<sup>20</sup>Il s'agit d'un algorithme probabiliste structuré de manière synchrone en rounds. Ainsi chaque sommet actif  $v$  est ajouté selon une probabilité  $P[v]$  relative au nombre de voisins actifs, et donc de son degré, avec  $P[v] = 1/2\deg[v]$ . Si deux voisins  $u$  et  $w$  sont sélectionnés dans le même round  $t$ , alors le sommet de degré minimal est retiré de  $\mathcal{U}^{[t]}$ . Une fois le stable  $\mathcal{U}^{[t]}$  extrait à l'itéré  $t$ , on construit le sous-graphe  $G^{[t]} \subset G^{[t-1]}$  induit par  $V^{[t]}$  privé de  $\mathcal{U}^{[t]} \cup \mathcal{N}[\mathcal{U}^{[t]}]$ , et on réitère la procédure jusqu'à ce qu'il n'y ait plus de sommets à traiter, c'est à dire lorsque  $|V^{[t]}| = 0$ .

<sup>21</sup>Elle se termine en  $O(\log n)$  itérés avec  $O(m)$  cores.

<sup>22</sup>Le sous-graphe  $G^{[t]} = (V^{[t]}, E^{[t]})$  est induit par  $V^{[t-1]}$  privé de  $\mathcal{U}^{[t]} \cup \mathcal{N}[\mathcal{U}^{[t]}]$  à chaque itéré  $t$ . Ici l'extraction des arêtes  $E^{[t]} = \{(u, v) \in E^{[t-1]} \text{ tq } u, v \in \mathcal{U}^{[t]}\}$  implique une intersection ensembliste et donc de multiples comparaisons et recopies de données.

<sup>23</sup>En fait  $q$  est calculée de sorte que  $n^{[t]} \leq q \leq 2n^{[t]}$  avec  $n = |V^{[t]}|$ .

<sup>24</sup>Les deux problèmes sont mutuellement réductibles : calculer une coloration permet d'extraire un stable et inversement.

<sup>25</sup>c'est-à-dire structuré en rounds ou itérés.

<sup>26</sup>Elle consiste à rajouter successivement chaque sommet  $v \in V^{[t]}$  dans  $\mathcal{U}^{[t]}$  tout en supprimant ses voisins  $\mathcal{N}[v]$  des sommets restants à traiter  $V^{[t]}$ . La taille du stable  $\mathcal{U}^*$  extrait vaut au moins  $\lceil \frac{n}{\Delta+1} \rceil$  où  $\Delta$  est le degré du graphe.

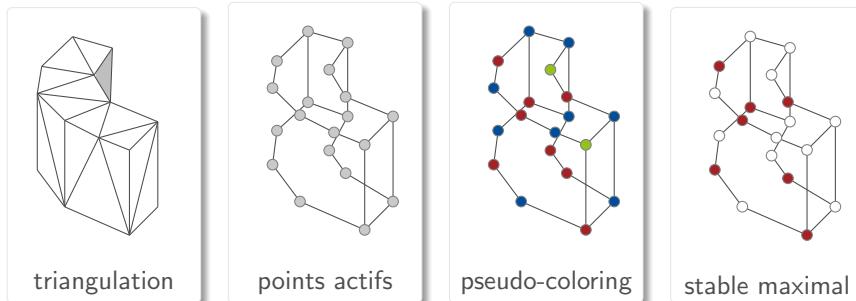


Figure 4.7: Extraction de tâches pour la simplification.

**PRINCIPE.** À l'issue du profiling, nous décidons de construire notre méthode d'extraction de stable à partir de l'approche spéculative de coloration de CATALYUREK [204] dont le principe est décrit à la figure 4.7.

Partant du constat que le stable de taille maximale  $\mathcal{U}^*$  fournie par le noyau First-fit coïncidait toujours avec le stable de plus petite couleur, nous décidons de résoudre les conflits uniquement pour cette couleur afin d'accélérer cette phase. Partant également du constat que l'approche MONTE-CARLO fournissait un meilleur ratio de  $|\mathcal{U}^*|$  sur  $|V|$  puisqu'il tenait compte du degré des points (voir page 150), nous décidons de garder le point de faible degré en cas de conflit, et de remettre le point non choisi dans la liste des tâches à traiter au prochain itéré, contrairement à [204]. Notons que la détection de points defectueux et leur résolution sont effectués dans deux phases synchrones distinctes (contrairement à [206]) afin d'éviter l'effet SIMD sur les processeurs récents avec plusieurs unités vectorielles<sup>27</sup>. Enfin la routine complète est décrite à l'algorithme 4.1.

**Algorithme 4.1:** Extraction de points pour la simplification.

```

initialiser  $V^{[t]}$  à liste des points à supprimer.
répéter
    pour chaque point  $p$  de  $V^{[t]}$  faire en parallèle
        pour chaque voisin  $q$  de  $p$  faire
            assigner  $\text{forbidden}[\text{tid}, \text{color}[q]] = p$ 
            assigner  $\text{color}[p] = \min\{c > 0, \text{forbidden}[c] \neq p\}$ 
    barrière
    pour chaque point  $p$  de  $V^{[t]}$  faire en parallèle
        si  $\text{color}[p] = 1$  et s'il y a un voisin  $q$  tq  $\text{color}[q] = 1$  alors
            si  $\deg[p] \geq \deg[q]$  alors
                rajouter  $p$  dans  $\ell_{\text{tid}}$ 
            fin si
        fin si
         $n_{\text{tid}} = |\ell_{\text{tid}}|$  et  $n^{[t+1]} = |V^{[t+1]}|$ 
        déterminer un offset par un prefix-sum de  $n^{[t+1]}$  avec  $n_{\text{tid}}$ .
        vider le contenu de  $\ell_{\text{tid}}$  dans  $V^{[t+1]}$  à cet offset.
    barrière
     $t = t + 1$ .
jusqu'à  $V^{[t]} = \emptyset$ 

pour chaque point actif  $p$  faire en parallèle
    rajouter  $p$  dans  $\ell_{\text{pid}}$  si  $\text{color}[p] = 1$ 
barrière
retourner  $\mathcal{U}^* = \bigcup_{k=1}^p \ell_{\text{tid}}$ .
```

<sup>27</sup>telles que l'Intel Knights Landing avec 4 VPU par core.

### 4.2.1.3 Pour la relaxation

En fait la stratégie basée sur l'extraction d'un stable pour identifier les tâches non conflictuelles est suffisamment générique pour être appliquée à tous les noyaux. Pour cela il faudrait juste bien formuler les dépendances de tâches en un graphe adéquat pour chaque noyau. Pour la relaxation par exemple, nous aurions eu besoin d'identifier un ensemble d'arêtes disjointes à ordonner, puisque chaque tâche correspond à la bascule d'une arête dans ce cas. Néanmoins nous décidons d'extraire les tâches d'une autre manière pour ce noyau. Au lieu de considérer les arêtes, nous considérons les paires de mailles. Pour trouver les arêtes à basculer, nous calculons un **couplage** du sous-graphe des mailles actives induit par le graphe dual de la triangulation.

**RAISONS.** Dans une quête perpétuelle de localité, nous avions eu recours à une **carte combinatoire**<sup>28</sup>. Avec cette représentation, les mises à jour des données d'incidence sont locales aux deux mailles incidentes à l'arête à basculer<sup>29</sup>, contrairement à un graphe d'incidence (figure 4.8). Dans ce cas, l'idée serait de construire un graphe où chaque sommet correspond à une arête de la triangulation, et chaque arête exprime le fait que deux arêtes de la triangulation sont adjacentes. Ainsi pour obtenir  $S$ , il nous aurait suffi d'extraire un stable de  $G$  et donc de réutiliser la méthode décrite à l'algorithme 4.1.

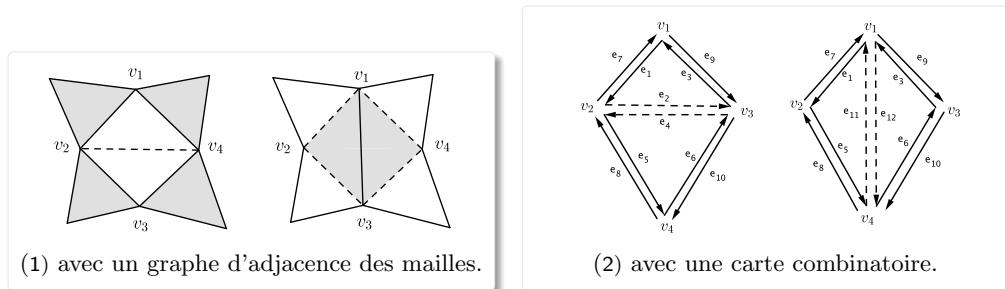


Figure 4.8: Voisinage impacté par la bascule d'arêtes.

Le problème avec la **carte combinatoire** est qu'elle implique d'importants défauts de cache induites par les requêtes de voisinage. C'est particulièrement le cas de la simplification dans la mesure où les indirections mémoire impactaient directement le scaling de ce noyau. Ainsi nous avons opté pour un graphe d'incidence où chaque point stocke les références des mailles qui lui sont incidentes. En termes d'accès-mémoire, cela nous permet de reconstruire rapidement le voisinage d'un point car on a beaucoup moins d'indirections. Pour la relaxation, construire le graphe de tâches  $G$  devenait néanmoins trop coûteux car on a plus une représentation explicite des arêtes : en fait la stratégie initiale n'était plus viable. En recherche d'une autre alternative, nous avons considéré les **paires de mailles** au lieu des arêtes. Ainsi l'idée est donc de construire  $G$  à partir du **graphe dual** puis d'extraire l'ensemble des arêtes à basculer en calculant un couplage de  $G$  comme sur la figure 4.6.

**TRAVAUX CONNEXES.** Le couplage est aussi un problème irrégulier [212]. Presque 80% des travaux récents [183, 213–218] se concentrent sur les graphes bipartis aussi bien pour le couplage de cardinalité ou de poids maximal, puisqu'ils sont conceptuellement plus simples à gérer. En recherche d'alternatives viables pour le cas non-biparti, nous nous sommes intéressés aux stratégies inhérentes aux approches, notamment celles basées sur la recherche de chemins augmentants [183, 215, 216]<sup>30</sup>. Pour atténuer la forte irrégularité du problème, les auteurs préconisent une approche de **codesign** qui vise à construire

<sup>28</sup>Il s'agit d'une structure de données topologique dans laquelle les relations d'adjacence des arêtes de la triangulation étaient exprimées explicitement (voir définition 15, page 27).

<sup>29</sup>En effet, la mise à jour des données d'incidence n'implique que les demi-arêtes relatives aux deux mailles  $K$  et  $R$  incidentes à l'arête comme sur la figure 4.8), tandis qu'elle aurait impliqué les quatre mailles voisines à  $K$  et  $R$  avec un graphe d'adjacence des mailles.

<sup>30</sup>De manière synthétique, elles sont basées sur une extraction multi-source de chemins augmentants par le biais d'une recherche en largeur (BFS) et/ou en profondeur d'abord (DFS) ainsi qu'au recours aux synchronisations à granularité fine (verrous, queue-based etc.).

un algorithme sur mesure pour une architecture cible<sup>31</sup>[212]. Sans rentrer dans leurs spécificités, nous avons identifié et résumé les points-clés d'un portage efficient de ces approches à la table 4.1.

Table 4.1: Comparaison des approches basées sur la recherche de chemins augmentants.

critères	DFS-based [183, 216, 218]	BFS-based [215, 216]
longueur chemins	assez long	court
parallélisation	par sommet source	par niveau
synchronisation	à chaque itéré (grain moyen)	à chaque niveau (grain fin)
facteur limitatif	déséquilibre de charges	surcoût des synchronisations
irrégularité	plus sensible à l'ordonnancement	moins sensible à l'ordonnancement
convergence	plus d'itérés peu coûteux	peu d'itérés plus coûteux
scaling	bonne strong scaling	$t_{\max}$ invariant; bonne weak scaling.
architecture	manycore	massivement multithread (GPGPU)

En fait ces approches sont des noyaux de **recherche locale** : elles procèdent par amélioration ou complétion d'un couplage initial  $\mathcal{U}^{[0]}$ . Pour construire le couplage  $\mathcal{U}^{[t]}$ , elles se basent sur des approches gloutonnes à l'instar de **greedy**. À l'itéré  $t$ , elle consiste en un parcours de  $G^{[t]}$  en choisissant de manière arbitraire, ou selon un critère spécifique basé sur les degrés, une arête à rajouter dans  $\mathcal{U}^{[t]}$  tout en la retirant de  $G^{[t]}$ . Les variantes se distinguent sur la manière de choisir l'arête en question<sup>32</sup>

**PRINCIPE.** À vrai dire, fournir une version parallèle à **moindre coût** d'une heuristique de recherche locale pour les **graphes non-bipartis** est réellement ardue : les performances des méthodes existants sont décevantes en pratique (sec. B.2.2). Ainsi nous nous sommes tournés vers les noyaux gloutons. À ce titre nous avons fourni un portage multithread et dérandomisée de **greedy**, en tenant compte des aspects décrits à la table 4.1. À l'itéré  $t$ , l'idée est de saturer incrémentalement le couplage  $\mathcal{U}^{[t-1]}$  en construisant les chemins alternants par un **parcours en profondeur** à partir d'un ensemble de sommets sources noté  $S$  comme sur la figure 4.9. Notons que l'approche est à grain fin car chaque sommet source  $s$  de  $S$  peut être attribué à un thread qui évolue de manière indépendante, et les synchronisations se font au niveau des sommets visités sont marqués par le biais de primitives atomiques **compare\_and\_swap** et **fetch\_and\_sub** (on n'utilise donc pas de barrière).

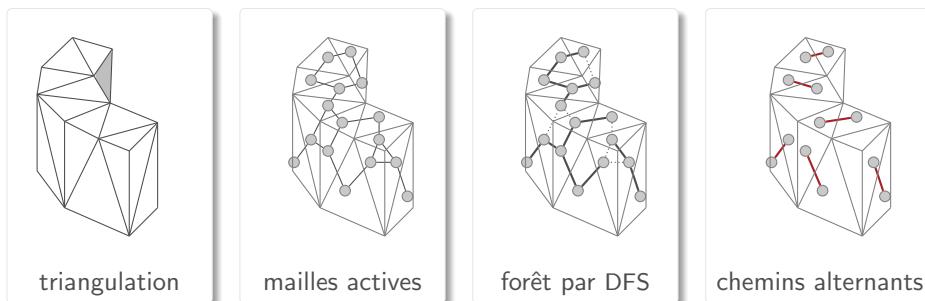


Figure 4.9: Couplage de mailles par recherche multi-source de chemins alternants.

De manière concrète, chaque thread va maintenir une pile locale de sommets sources  $S$ , et pour chaque  $s \in S$  il va parcourir son voisinage à distance deux, en rajoutant l'arête  $(s, u)$  dans  $\mathcal{U}^{[t]}$  si le voisin  $u$  de  $s$  n'a pas encore été visité par un autre thread. En nous inspirant des approches de KARP-SIPSER et min-degree et étant donné que  $\Delta = 3$ , nous décidons de rajouter en priorité le prochain voisin  $v \in N[u]$  de degré un dans la pile  $S$  pour accélérer la procédure. Pour cela, le degré de chaque sommet candidat  $w$  est décrémenté à chaque visite, et  $w$  n'est rajouté que si  $\deg[w] = 1$ . La recherche est ensuite effectuée jusqu'à ce que  $S$  soit vide comme décrit à l'algorithme ??.

<sup>31</sup>En fournissant une version **dataflow** et en recourant aux primitives natives à la machine pour la synchronisation.

<sup>32</sup>On distingue notamment celle de KARP-SIPSER auquel cas des sommets de degré 1 auquel cas l'arête est automatiquement ajoutée dans  $\mathcal{U}^{[t]}$ , des autres sommets auquel cas l'arête est rajoutée selon une probabilité.

**Algorithme 4.2: Couplage de mailles pour la relaxation.**

```

initialiser  $\mathcal{U}^* = \emptyset$  et  $\text{visited}[i] \leftarrow 0, \forall i$ .
pour chaque maille  $s$  faire en parallèle
     $S[\text{tid}] = \{s\}$ .
    répéter
        dépiler  $u = S[\text{tid}][0]$ 
        si  $\text{atomic\_compare\_swap}(\text{visited}[u])=0$  alors
            pour chaque maille  $v$  voisine de  $u$  faire
                si  $\text{atomic\_compare\_swap}(\text{visited}[v])=0$  alors
                    rajouter  $(u, v)$  dans  $\mathcal{U}^*$ .
                pour chaque maille  $w$  voisine de  $v$  faire
                    si  $\text{atomic\_fetch\_sub}(\text{deg}[w], 1)=2$  alors
                        push  $w$  dans  $S[\text{tid}]$ .
                    fin si
                fin si
            fin si
        fin si
    jusqu'à  $S[\text{tid}] = \emptyset$ 
    barrière
    retourner  $\mathcal{U}^*$ 

```

### 4.2.2 Restructuration des accès-mémoire

PROBLÈME. À ce point, nous disposons d’algorithmes pour extraire le parallélisme amorphe inhérent à chaque noyau. Les tâches extraites sont suffisamment fines pour laisser à l’ordonnanceur la latitude nécessaire au rééquilibrage dynamique de charges<sup>33</sup>. Ainsi il nous reste à gérer l’aspect data-intensive. Pour améliorer la réutilisation des données en cache et minimiser la latence des accès-mémoire, il faudrait restructurer les insertions et suppressions de données de chaque noyau. En fait ces indirections mémoire peuvent impacter de manière significative les performances de ces noyaux<sup>34</sup> surtout s’ils ont une faible intensité arithmétique<sup>35</sup>. Le problème est qu’il nous est impossible de les inférer puisque les dépendances de données varient selon la manière dont la topologie évolue<sup>36</sup>. Dans ce cas, comment allons nous débrouiller pour restructurer nos accès-mémoire de manière à atténuer les pénalités relatives à ces indirections ?

#### 4.2.2.1 Refonte en vagues synchrones

PRINCIPE. Afin d’atténuer leur irrégularité, nous restructurons les noyaux en **vagues synchrones** en nous inspirant du paradigme MULTI-BSP<sup>37</sup>[170]. Ici les instructions d’un noyau sont structurés en une séquence de vagues de tâches, chaque vague étant constituée d’une phase de **calcul local**, d’une phase de **communication** et d’une barrière de **synchronisation** comme montré à la table 4.2. Ici l’avantage est triple :

- **coalescence** : les communications des threads sont regroupées en vagues de manière à atténuer la latence mémoire<sup>38</sup>. Les écritures en mémoire partagée sont ainsi effectués de manière coalecente, ce qui est un aspect crucial sur les architectures massivement multithread<sup>39</sup>.

<sup>33</sup>à condition d’ajuster la granularité

<sup>34</sup>C'est particulièrement vrai pour la simplification comme illustré sur la courbe de la figure B.9 en annexe.

<sup>35</sup>L'intensité arithmétique est le ratio de calcul utile sur le nombre d'accès de données. Elle est mesurée en FLOP/BYTE

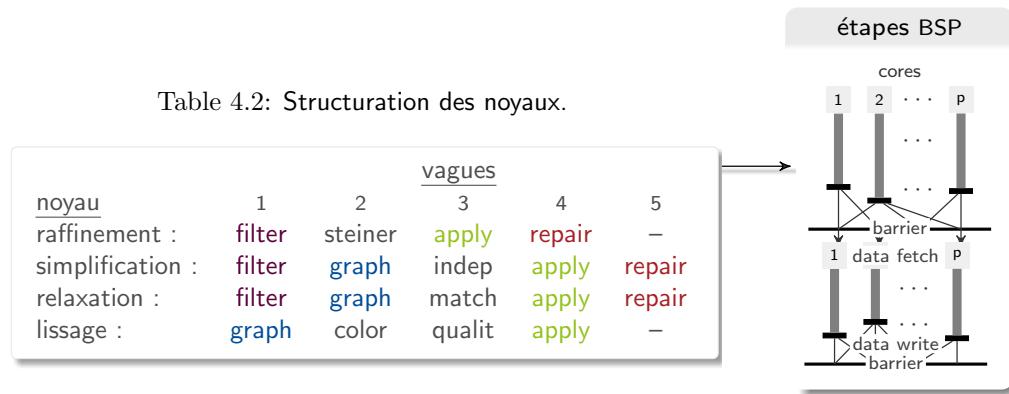
<sup>36</sup>Du coup, nous ne pouvons pas recourir aux techniques usuelles de **cache blocking**, ni préserver un placement de données **cache-aware**, comme ce qui se fait en visualisation haute performance [57, 58]

<sup>37</sup>Il s’agit d’un **bridging-model** entre le matériel et l’algorithme : il permet de concevoir des algorithmes tenant compte des paramètres hardware liés à la communication (bande-passante et latence à chaque niveau de la hiérarchie mémoire) tout en restant suffisamment générique pour la portabilité de l’application.

<sup>38</sup>Plus précisément, elles sont coalescées par thread et effectués dans une seule phase d’une vague synchrone.

<sup>39</sup>C'est particulièrement vrai pour le GPGPU.

- moins de communications : elle permet de réduire la fréquence de mise à jour de données entre threads ainsi que les synchronisations associées<sup>40</sup>.
- portabilité : elle offre un juste milieu entre prise en compte des contraintes hardware et générativité pour la portabilité des performances. De plus, elle fournit un modèle de coût lié aux paramètres hardware, ce qui peut-être utile pour le rééquilibrage de charges<sup>41</sup>



**SYNTHESE.** Les vagues synchrones relatives à chaque noyau sont résumées à la table 4.2. Les détails de décomposition des noyaux ainsi que leurs complexités théoriques<sup>42</sup> sont expliqués dans [RLP16]<sup>43</sup>. Sur cette table, les vagues

- filter correspond au filtrage des points ou mailles actives selon un critère numérique.
- graph correspond à la construction du graphe de conflits ou mailles à appairer.
- apply correspond à l'application proprement dite du noyau.
- repair correspond à l'épuration des listes d'incidence inconsistantes.
- steiner correspond au calcul et résolution des indices de points de STEINER.

En fait tout l'intérêt de ce formalisme réside dans la coalescence des communications<sup>44</sup> des threads, ce qui permet de minimiser les synchronisations tout en atténuant la latence des accès-mémoire. Bon nombre de bibliothèques existent pour rendre ces communications transparentes [171–174]. Néanmoins cela est inadapté dans notre cas, puisque nous voulons contrôler finement la manière dont les transferts de données sont effectués en mémoire partagée.

#### 4.2.2.2 Insertions de mailles

Pour le raffinement, nous disposons de plusieurs patterns de découpage selon le nombre d'arêtes "longues" conformément à une métrique donnée. Ainsi il est a priori impossible de prédire en amont le nombre de cellules à insérer. Pour y remédier, une manière simple serait de les stocker localement jusqu'à ce que chaque thread ait terminé de traiter toutes les mailles qui lui ont été assignées, puis de les copier de manière synchronisée au sein du conteneur partagé dans une seule vague, comme ce qui est implémenté dans Pragmatic [199, 167]. Le problème est que cela induit un nombre important d'accès-mémoire, ce qui peut être rédhibitoire en contexte NUMA.

**PRINCIPE.** Pour contourner le problème précédent, nous scindons le noyau en deux phases synchrones relatives au filtrage et à l'application, de sorte à pouvoir inférer explicitement le nombre de cellules à insérer. Ainsi cela va nous permettre de trouver le bon offset d'indices par thread.

Concrètement, nous stockons le pattern par maille à appliquer dans un tableau **pattern** durant la phase de filtrage. Ensuite, chaque thread  $t_i$  effectue une réduction sur **pattern** dans son espace d'itérations  $[n/p][i, i + 1]$ , avec  $n$  le nombre de tâches et  $p$  le nombre de threads. Le résultat est

<sup>40</sup>Elle minimise ainsi les contentions d'accès aux conteneurs partagés, typiquement les files de tâches associé à chaque noyau et la structure de données topologiques

<sup>41</sup>Dans ce cas, elle permet de décider s'il faut migrer les mailles ou non selon le coût estimé du prochain itéré.

<sup>42</sup>à l'aide du modèle de pont Queuing Shared Memory.

<sup>43</sup>dans le cas planaire mais c'est exactement pareil en surfacique

<sup>44</sup>par mise à jour synchronisée de variables en mémoire partagée.

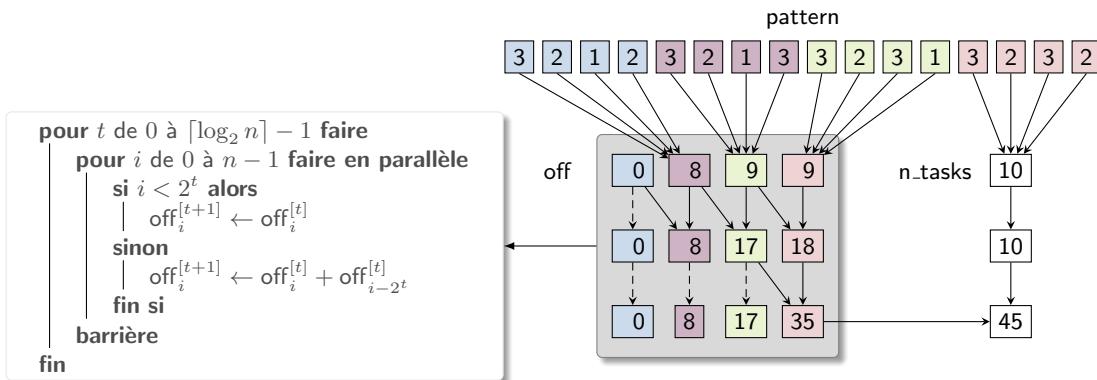


Figure 4.10: Précalcul des références par thread pour l'insertion de mailles. Les motifs de raffinement associés à chaque maille sont explicitement référencés dans un tableau, et chaque thread détermine la quantité de mailles qu'il doit créer par une réduction partielle. Enfin un prefix-sum est effectué pour déterminer les offsets associés à chaque thread.

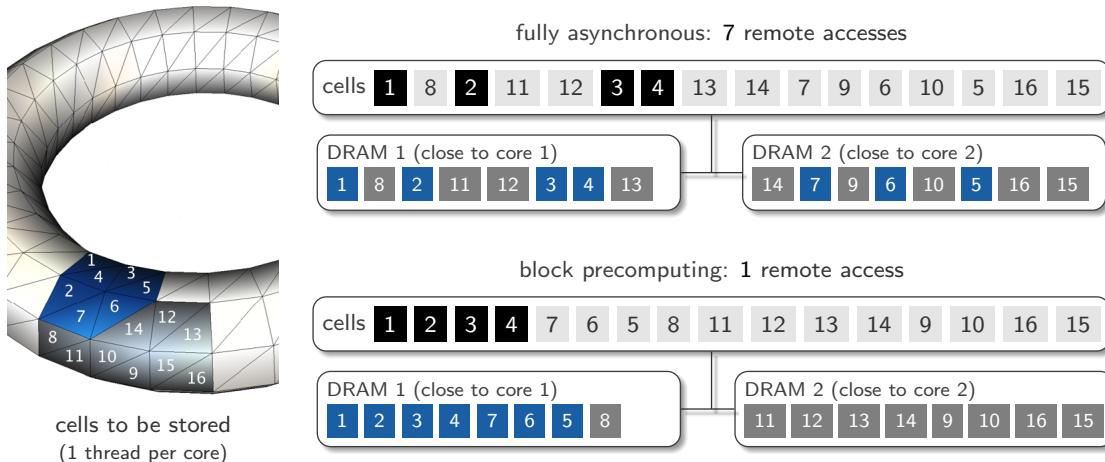


Figure 4.11: Impact des patterns d'insertion de mailles sur le placement mémoire en contexte NUMA.

ensuite stocké dans un tableau  $\text{offset}[i]$  de taille  $p$ . Finalement, un prefix-sum est effectué sur  $\text{offset}[i]$  afin de déterminer les plages d'indices  $[k_i, k_{i+1}]$  par thread.

L'exemple de la figure 4.11 illustre l'impact du pattern d'insertion de mailles sur le placement mémoire. Ici on a deux threads punaisés sur deux cores situés sur deux sockets distincts. Dans le premier cas, les indices des mailles sont obtenus de manière asynchrone et les threads insèrent directement dans le conteneur partagé. Dans le second cas, les plages de blocs-mémoire sont pré-déterminées dans une première vague, et les insertions sont réalisées de manière coalescente dans une vague à part. Dans ce cas, les blocs de données associés à chaque thread sont réellement stockés dans la mémoire la plus proche du core d'une part, et les mailles voisines géométriquement le sont également en mémoire.

#### 4.2.2.3 Réduction de données

À ce point, chaque noyau est structuré de manière à ce que la communication des threads se fasse de manière coalescente. De manière concrète, elle consiste en une mise à jour synchronisée de données en mémoire partagée. En fait les points et mailles ainsi que les données afférentes sont stockées dans des tableaux unidimensionnels de sorte qu'elles sont référencées par des adresses contigües en mémoire. Dans ce cas, les primitives d'insertion, de suppression ou de mise à jour des conteneurs

partagés doivent être synchronisées. Ici le but est de montrer comment les réductions<sup>45</sup> de données sont réellement effectués par les threads.

**RÉDUCTIONS.** Les boucles de work-sharing impliquées dans chaque vague synchrone nécessitent le recours à un mécanisme de réduction de données au sein d'une file de tâches ou d'un conteneur partagé. Ici les deux points critiques concernent la minimisation des points de synchronisations et la préservation du placement mémoire initial. En fait il est compliqué de concilier les deux : l'asynchronisme implique l'insertion non déterministe des données tandis que le placement fin des données implique nécessairement plus de points de synchronisations. Partant de constat, nous proposons deux stratégies de réduction. Elles sont basées sur la pré-détermination des offsets  $\text{off}[\text{tid}]$  à partir desquels chaque thread  $\text{tid}$  peut initier sa copie de données dans le conteneur partagé  $\text{R}$ . Ces deux stratégies sont :

- ASYNCHRONE : ici  $\text{off}[\text{tid}]$  est calculé de manière non-déterministe à partir de la taille  $n_{\text{R}}$  du conteneur partagé  $\text{R}$ . À l'instant  $t$ ,  $n_{\text{R}}^{[t]}$  est incrémenté de manière atomique tout en récupérant en cache son ancienne valeur  $n_{\text{R}}^{[t-1]}$  qui sera assignée à  $\text{off}[\text{tid}]$  : ce mécanisme s'appelle la **capture atomique**. Ainsi le thread  $\text{tid}$  sait exactement qu'il doit copier ses données aux indices  $[n_{\text{R}}^{[k-1]}, n_{\text{R}}^{[k-1]} + |\ell_{\text{tid}}|]$ , où  $\ell_{\text{tid}}$  désigne sa liste locale de données. Notons qu'on a bien un **asynchronisme** puisque le thread  $\text{tid}$  n'attend pas que la réduction se termine pour faire du calcul local. Par contre, la plage d'adresses associée à  $\text{tid}$  est complètement arbitraire et varie d'une exécution à une autre.
- NUMA-AWARE : cette fois  $\text{off}[\text{tid}]$  est mis à jour par le biais d'un **prefix-sum** de sorte que les plages d'adresses soient assignées de manière déterministe aux threads. En fait elles dépendent de l'indice  $\text{tid}$  du thread qui lui est punaisé statiquement sur un core : cela permet ainsi d'allouer une plage d'adresses la plus proche possible de ce core. Notons qu'ici on a  $\log(p)$  réductions et donc autant de points de synchronisation.

Listing 4.1: quasi-asynchrone

```
void trigen::fast_reduce(int tid,
                        std::vector<int>* heap,
                        int* shared,
                        int* off,
                        int* size)
{
    int nb = heap[tid].size();
    off[tid] = __sync_fetch_and_add(size, nb);
    // tasks:null si on veut juste calculer les offsets
    if(shared != nullptr)
        memcpy(shared+off[tid],
               heap+tid, nb*sizeof(int));
}
```

Listing 4.2: NUMA-aware

```
void trigen::numa_reduce(int tid,
                        std::vector<int>* heap,
                        int* shared,
                        int* off)
{
    int nxt = (tid+1)%n_cores;
    off[nxt] = heap[tid].size();
    #pragma omp barrier
    prefix_sum(heap, off);
    if(shared != nullptr)
        memcpy(shared+off[tid],
               heap+tid, off[nxt]*sizeof(int));
}
```

Notons que ces mécanismes de réduction n'impliquent que des variables entières ( $\text{heap}$ ,  $\text{shared}$ ). Pour minimiser le volume de données transférées en mémoire partagée, les réductions ne sont utilisées que pour la copie d'indices. Ainsi les cellules et les données qui leur sont associées (normales, tenseurs etc.) sont directement créés et initialisées à leur emplacement final, contrairement à l'approche de Rokos [199]. Étant donné leur caractère data-intensif, ces multiples transferts de données impactent les performances des noyaux de manière significative, notamment en 2D (sec 4.2.3).

### 4.2.3 Consistance des données d'incidence

À ce point nous disposons de moyens pour extraire le parallélisme amorphe inhérent à chaque noyau. En fait nous n'avons géré que les conflits de tâches relatives à la **conformité** de la topologie,

<sup>45</sup>Une réduction est une agrégation de données locales aux processus.

afin de maximiser le nombre de tâches extraites en vue d'alimenter suffisamment les cores. Ainsi il nous reste à gérer explicitement la consistance des données d'incidence lorsqu'elles sont mises à jour de manière concurrente. Rappelons que la topologie est stockée et maintenue dans les listes d'incidences (page 58). Maintenant, on vise à définir une synchronisation lock-free pour la mise à jour concurrente de ces listes de manière à minimiser les transferts de données.

#### 4.2.3.1 Contraintes en lock-free

APPROCHES. En fait fournir une stratégie lock-free et non coûteuse pour la mise à jour des données d'incidence n'est pas trivial. Le problème est souvent résolu de deux manières :

- **ÉVITEMENT.** Le recours à une structure de données orientée de type carte combinatoire<sup>46</sup> permet d'éviter ce problème, comme énoncé à la section 4.2.1.3. En fait les mises à jour restent locales au sous-ensemble de mailles en cours de modification, et aucune synchronisation n'est nécessaire dans ce cas. Néanmoins elle induit un nombre important d'indirections mémoire lors des requêtes de voisinage<sup>47</sup> : cela impacte directement au scaling des noyaux, notamment pour la simplification (page 153). Une autre alternative possible est de prendre en compte les mises à jour du graphe d'incidence au moment de l'extraction des tâches. Dans [210] par exemple, FREITAG contourne le problème grâce à une 2-coloration de graphes. Néanmoins elle est plus coûteuse et requiert plus de couleurs que le cas classique : comme la partition obtenue est plus grande, la taille des stables est réduite. Ainsi le nombre de tâches extraites par itéré est réduit, tandis que le nombre d'itérés nécessaire pour converger augmente. En effet le nombre chromatique  $\chi_{2,G}$  est plus grand que  $\chi_{1,G}$  et croît selon le degré maximal  $\Delta$  de  $G$ . Plus précisément, si la triangulation est :
  - isotrope alors on a  $\chi_{1,G} \leq \chi_{2,G} \leq \Delta + 5$ , avec un degré max  $\Delta \approx 6$ .
  - anisotrope alors on a  $\chi_{1,G} \leq \chi_{2,G} \leq \frac{3}{2}\Delta$  avec  $\Delta \geq 8$  (preuve dans [201]).
- **GESTION DIFFÉRÉE.** Une autre solution consiste à stocker localement les mises à jour à chaque itéré, puis de procéder à une réduction dans le conteneur partagé en fin d'itéré. Dans [199, 167] par exemple, ROKOS fournit une alternative lock-free pour la mise à jour du graphe d'incidence. À l'itéré  $t$ , elle consiste à reporter la copie des données topologiques en fin du traitement des tâches de  $\mathcal{U}^{[t]}$ , ce qui garantit d'utiliser des données valides puisque la topologie est figée à cet instant. Pour cela, il dispose d'une matrice de listes de mise à jours  $\text{def}$  décrite à (4.1).

$$\begin{array}{c|ccccccccc}
 & & & & \text{updates} & & & & \\
 & & & & t_0 & t_1 & t_2 & \cdots & t_p \\
 \text{commits} | & t_0 & \left[ \begin{array}{cccccc} \text{def}[0][0] & \text{def}[1][0] & \text{def}[2][0] & \cdots & \text{def}[p][0] \\ \text{def}[0][1] & \text{def}[1][1] & \text{def}[2][1] & \cdots & \text{def}[p][1] \\ \text{def}[0][2] & \text{def}[1][2] & \text{def}[2][2] & \cdots & \text{def}[p][2] \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ \text{def}[0][p] & \text{def}[1][p] & \text{def}[2][p] & \cdots & \text{def}[p][p] \end{array} \right] & & & & & & & (4.1) \\
 t_1 & & & & & & & & \\
 t_2 & & & & & & & & \\
 \vdots & & & & & & & & \\
 t_p & & & & & & & & 
 \end{array}$$

Ici, on distingue deux vagues d'opérations :

- **updates** : lorsque un thread  $i$  doit mettre à jour le voisinage d'un point, il copie la liste partielle dans celle du thread indexé par  $j = i \bmod p$ , avec  $p$  le nombre total de threads.
- **commits** : à l'issue du traitement de  $\mathcal{U}^{[t]}$ , chaque thread  $i$  parcourt le tableau de listes de chaque thread  $k \in [1, p]$ , puis repère la liste  $\text{def}[k][i]$  qui lui a été réservée. Ensuite il transfère chaque liste  $\ell \in \text{def}[k][i]$  dans le conteneur associé au graphe  $(P, M, n)$ .

In fine les listes partielles  $\{\ell_{i,j}\}_{i=1,p}$  relative à un point  $p_j$  sont bien copiées par un unique thread : il n'y a donc pas de data races. L'inconvénient de cette stratégie est qu'elle engendre énormément de transferts et recopies de données, ce qui est réellement critique dans notre contexte.

<sup>46</sup>Dans ce cas, la topologie est représentée par les relations d'adjacence des demi-arêtes (`next`, `twin`), voir page 27.

<sup>47</sup>boucle d'un point, mailles voisines, coquilles d'une arête par exemple.

### 4.2.3.2 Notre synchronisation

PRINCIPE. Partant de ces constats, nous proposons une mise à jour synchronisée en deux temps. Rappelons qu'ici, chaque point garde les références des mailles qui lui sont incidentes. Ici, nous distinguons les opérations d'insertions et de suppressions de références dans la liste d'incidence  $\text{incid}[p]$  de chaque point  $p$ . Pour chaque noyau, l'idée est de permettre aux threads de rajouter les références de manière asynchrone dans une vague à part. Pour cela, les threads incrémentent de manière atomique le degré  $\text{deg}[p]$  du point qui est stocké explicitement sous forme d'un tableau. En effet, cela va permettre de déterminer les **offsets** des listes d'incidences de manière asynchrone : chaque thread met à jour  $\text{incid}[p]$  et poursuit son chemin. Comme sa liste d'incidence peut contenir des références obsolètes, le point est ensuite marqué comme étant à réparer, et cela de manière atomique par le biais d'un flag  $\text{fix}[p]$ . Enfin quand tous les threads ont terminé leurs modifications, les listes d'incidence de chaque point marqué sont réparées dans une vague à part comme illustré sur l'exemple de la figure 4.12.

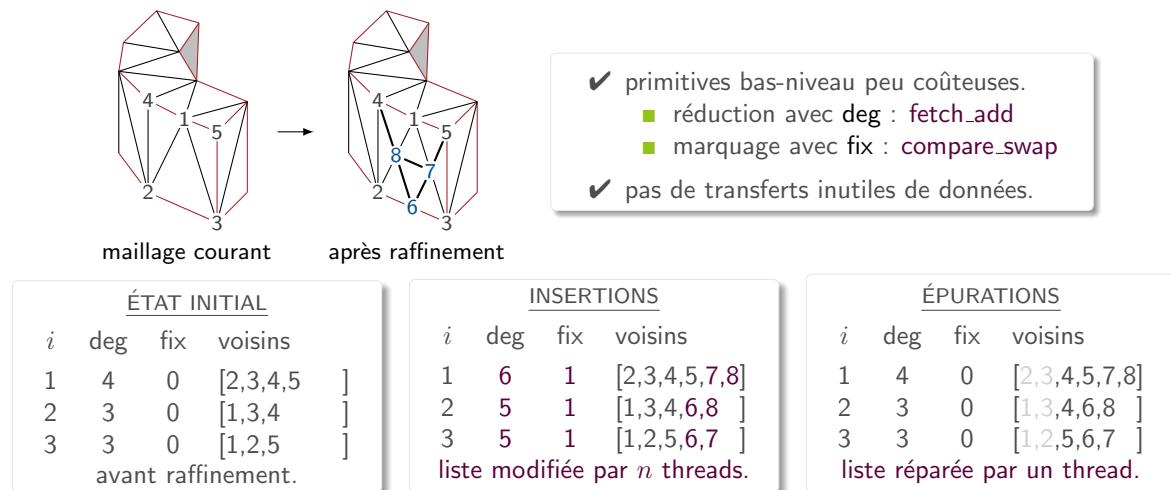


Figure 4.12: Mise à jour synchronisée en deux temps du graphe d'incidence.

Notons juste que les listes d'incidence ont une capacité fixe, et il se peut qu'un thread ne puisse plus insérer ses données. Après avoir calculé son offset, le thread vérifie si la taille des données à insérer excède cette capacité, auquel cas il réalloue la liste d'incidence en doublant sa capacité actuelle. Pour éviter une réallocation multiple (plusieurs threads), nous recourons au pattern singleton implémenté par un double checking (voir algorithme 4.3). En pratique, la mémoire allouée au graphe d'incidence est ajustée aux paramètres des noyaux (seuil sur les itérés, nombre de mailles créées etc.) de manière à minimiser ces réallocations. La routine complète est décrite à l'algorithme 4.3.

EFFICACITÉ. Comparée aux alternatives précédentes, notre stratégie présente quelques avantages :

- les primitives atomiques utilisées sont peu coûteuses<sup>48</sup>, et on a peu d'indirections contrairement à une carte combinatoire.
- elle préserve un degré de parallélisme élevé, car seuls les conflits de conformité sont considérées au moment de l'extraction de tâches, contrairement à l'approche de FREITAG [210].
- il induit un mouvement minimal de données contrairement à l'approche de ROKOS [199, 167]. En fait les threads mettent directement à jour les listes d'incidences au moment où elles doivent l'être, au lieu d'en garder une copie locale puis de procéder à une réduction des listes partielles au sein du graphe d'incidence.

\* \* \*

<sup>48</sup>Ils sont de l'ordre de 15-30 cycles CPU si le compteur est déjà en cache L1.

Algorithme 4.3: Primitives de mise à jour du graphe d'incidence.

```

fonction INSÉRER( $p_i, \ell_{\text{tid}}$ )
    atomic_compare_swap( $\text{fix}[i], 1$ )a
     $k = \text{atomic\_fetch\_add}(\text{deg}[i], n)$ b
    si  $n + k$  excède la capacité de  $\text{incid}[i]$  alors
        #critical           ▷ double check pattern
        si pas encore realloué alors
            doubler la capacité de  $\text{incid}[i]$ c.
        fin si
    fin si
    copier  $\ell_{\text{tid}}$  dans  $\text{incid}[i][k]$ .
fin

```

<sup>a</sup>fix : marqueurs des points à réparer.<sup>b</sup>deg : degré des points ⇔ offsets sur incid.<sup>c</sup>incid : listes d'incidence des points.

```

fonction RÉPARER( $p_i, \ell_{\text{tid}}$ )
    si  $\text{fix}[i]$ a alors
        pour chaque maille K de  $\text{incid}[i]$  faire
            si  $p_i$  référencé dans K alors
                ajouter K dans  $\ell_{\text{tid}}$ 
            fin
        vider  $\text{incid}[i]$ 
        réinitialiser  $\text{deg}[i] = |\ell_{\text{tid}}|$ b.
        trier  $\ell_{\text{tid}}$  and échanger avec  $\text{incid}[i]$ c.
    fin si
fin

```

<sup>a</sup>fix : marqueurs des points à réparer.<sup>b</sup>deg : degré des points ⇔ offsets sur incid.<sup>c</sup>incid : listes d'incidence des points.

## 4.3 ÉVALUATION NUMÉRIQUE

### 4.3.1 Cadre, architectures et paramètres

**RÉSUMÉ.** En bref, nous avons proposé une approche de parallélisation des noyaux adaptatifs dédié aux architectures multicore et manycore. Elle concilie les contraintes relatives à l’irrégularité<sup>49</sup> des noyaux avec celles du hardware, et toutes ses briques internes sont lock-free. Elle s’appuie sur :

- une extraction du parallélisme amorphe pour chaque noyau par le biais d’heuristiques de graphes. Afin d’alimenter suffisamment les cores, nous ne considérons que les conflits relatifs à la conformité de la topologie ce qui maximise le nombre de tâches extraites.
- une approche de restructuration des accès-mémoire irréguliers qui atténue la latence. Pour cela,
  - les noyaux sont structurés en vagues afin de réduire la fréquence des échanges de données.
  - les emplacements des mailles sont précalculés au moment de leur création afin d’éviter les recopies de données locales et pour préserver au mieux la proximité de mailles voisines.
  - les réductions de données se font de manière asynchrone ou NUMA-aware selon le contexte.
- une synchronisation à grain fin pour les mises à jour du graphe d’incidence. Elle est peu coûteuse et minimise les transferts et copies de données comparé à l’état de l’art.

Maintenant le but est de montrer expérimentalement l’efficacité de l’approche et des features proposées. Pour cela, elle a été implémentée en C++11 et utilise OpenMP4 pour le multithreading. Elle fait l’objet d’une bibliothèque baptisée **trigen** qui sera bientôt disponible en open-source. En fait nous avons initialement implémenté une version planaire de **trigen** avant de l’étendre au cas surfacique. Notons que l’approche est parfaitement adaptée aux deux cas, même si le profil de performances peut sensiblement varier car les noyaux surfaciques ont une intensité arithmétique un peu plus importante que leurs variantes planaires. À ce titre, nous tenons à présenter les deux cas ici.

**PLAN.** Pour commencer, nous fixons le cadre de nos benchmarks en décrivant les architectures et paramètres de tests. Ensuite nous évaluons l’efficience des briques que nous avons conçus pour l’extraction du parallélisme amorphe pour les noyaux de simplification et de relaxation. En particulier, nous montrons l’évolution du nombre de tâches extraites au fur et à mesure des itérés ainsi que leurs surcoûts sur le temps de restitution des noyaux. De même, nous évaluons notre schéma de synchronisation à grain fin pour la mise à jour des données d’incidence, en nous comparant avec celui de ROKOS en termes de surcoût induit. Après nous évaluons le scaling du remailleur et de chaque noyau sur des cas-tests planaires et surfaciques, et cela sur toutes les architectures. En particulier, nous montrons le surcoût induit par la parallélisation, et la manière dont il évolue quand le nombre

<sup>49</sup>de leurs aspects data-driven et data-intensive plus précisément, voir section 4.1.2

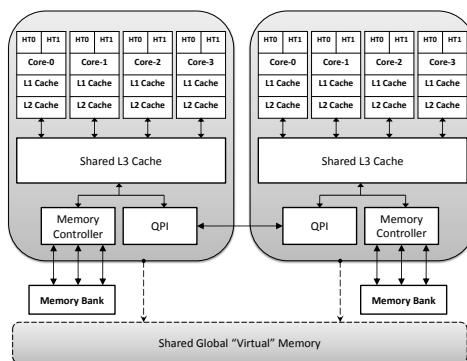
de cores utilisés augmente exponentiellement. Pour finir, nous profilons le débit de tâches traitées et la quantité de calcul utile pour chaque noyau.

Table 4.3: Caractéristiques des machines de tests.

code	puces	NUMA	cores	GHz	threads	GB/core	référence complète
HSW	2	4	32	2.5	64	4.0	Intel Xeon Haswell E5-2698 v3
SKL	2	2	48	2.7	96	7.9	Intel Xeon Skylake Platinum 8168
KNL	1	4	72	1.4	288	1.5	Intel Xeon-Phi Knights Landing 7250

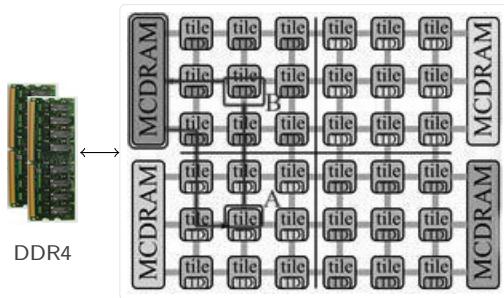
**ARCHITECTURES.** Le profiling a été effectué sur deux machines multicore et une machine manycore, dont les caractéristiques sont résumées à la table 4.3. Elles sont représentatives des noeuds de calcul que l'on trouve sur les clusters récents.

- **MULTICORE** : on a deux machines dual-socket (HSW, SKL) basées sur des processeurs Intel Xeon cadencés à 2.5 Ghz (turbo-boost à 3.4 Ghz) avec 3 niveaux de cache. HSW est une machine 32-core (avec 16 cores par puce) dont mémoire est structurée en quatre noeuds NUMA reliés en anneau, tandis que SKL est un noeud 48-core structuré en deux noeuds NUMA. Les deux machines disposent d'un cache L3 commun de 33 Mo par noeud NUMA.
- **MANYCORE** : on a une machine dual-memory (KNL) basée sur un processeur Intel Xeon-Phi cadencé à 1.4 Ghz. KNL a la particularité d'intégrer une mémoire on-chip MCDRAM avec une bande passante de 320 Go/s et une capacité fixe de 16 Go, ainsi qu'une mémoire classique DDR4 à 60 Go/s. Notons que la MCDRAM peut être utilisée en tant que cache supplémentaire ou comme une mémoire à part entière, en spécifiant le mode utilisé (**cache/flat**) au moment du boot. Afin d'optimiser l'utilisation des caches pour adapter au mieux avec les patterns d'accès-mémoire, les cores peuvent être logiquement regroupés en quadrant, en hémisphère (deux groupes de cores) ou en 4 domaines NUMA (mode **subnuma**). Ce mode devrait être privilégié afin de tirer profit du caractère **NUMA-aware** des noyaux<sup>50</sup>. Néanmoins, ne disposant pas de priviléges administrateurs sur le noeud de calcul, nous n'avons pu utiliser que le couple de modes **quadrant/flat**, ce qui revient à utiliser la machine en mode **Symmetric Multiprocessing**.



(1) hiérarchie de caches sur HSW et SKL

- un tile : deux cores et un cache-L2 partagé.
- un quadrant de six tiles : un noeud NUMA.



(2) clustering en quadrants sur KNL

Figure 4.13: Architecture mémoire de nos machines de tests.

<sup>50</sup>En effet la MCDRAM est répartie en quatre sur la puce, et les cores sont regroupés de sorte qu'ils soient physiquement plus proches de la MCDRAM intégrée au noeud NUMA. Ici les adresses physiques sont mappées aux tag directory de sorte que les transferts mémoire restent au maximum au sein du quadrant. Cela permet ainsi de réduire la latence en cas de défaut de cache tant que la donnée reste au sein du même noeud NUMA.

Afin de déterminer l'impact réel des défauts du dernier niveau de cache du processeur, nous avons profilé le débit et la latence mémoire effective sur les deux architectures (HSW et KNL) à la figure 4.14.

- pour la **bande-passante**, nous utilisons le célèbre benchmark **stream** [165] disponible sur <http://www.cs.virginia.edu/stream/> dont les résultats sont illustrés pour le noyau triad;
- pour la **latence**, nous utilisons le benchmark **lmbench** [175] disponible sur <http://www.bitmover.com/lmbench/> dont les résultats sont donnés pour les accès en lecture.

Pour KNL, le débit varie sensiblement selon qu'on utilise la MCDRAM ou la DDR4 (320 Go/s et 64 Go/s) mais la latence reste quasi-identique (30 ns et 28 ns). À l'inverse, le débit est identique pour l'accès à une mémoire locale ou distante pour HSW, mais la latence varie fortement selon que l'on accède à un bloc en cache (4.7 et 6.4 ns en cache L2-L3), à une mémoire locale (18 ns) ou distante (40 ns pour le noeud #4).

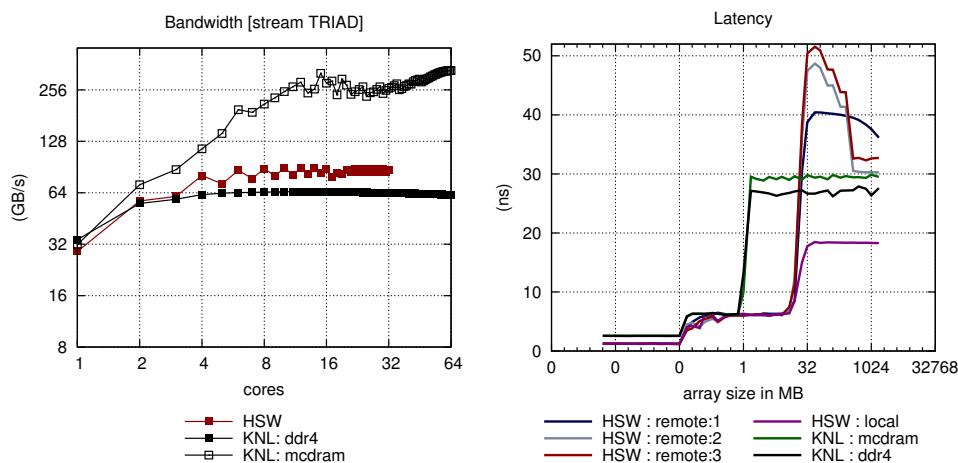


Figure 4.14: Débit et latence mémoire sur les deux architectures.

**PARAMÈTRES.** Le code de trigen a été compilé avec le compilateur d'Intel icpc avec le flag d'optimisation **-O3** et **qopt-prefetch=5** incluant l'auto-vectorisation et le prefetching logiciel. Afin de tirer profit des features spécifiques au hardware, nous activons les flags **-march=native** lors de la compilation sur HSW-SKL et **xmic-avx-512** sur KNL. Ici les threads sont explicitement punaisés sur les cores de manière compacte à raison d'un thread par core. Concrètement, cela est réalisé en positionnant la variable d'environnement **KMP\_AFFINITY=compact,granularity=unit** avec **unit =core|fine** en mode normal ou hyperthreading. En fait c'est le mode par défaut que nous utilisons sur KNL avec 4 HT par core comme recommandé par Intel. Pour les instances de tests, nous avons considéré les cas :

- 2D. Nous utilisons trois champs de solution à différents degré d'anisotropies pour nos tests, tels qu'illustrés à la figure 4.16. Pour chaque cas, nous utilisons une grille triangulée de 504 100 points et 1 005,362 mailles. Pour chaque run, une seule adaptation est effectuée sur trois itérés, et le facteur de résolution est fixé à 0.9<sup>51</sup>.
- SURFACIQUE. Nous considérons deux cas-tests : (1) une adaptation isotrope basées sur les courbures d'une pièce mécanique (**engine**) avec 1 826 000 points et 3 652 058 mailles, ainsi qu'(2) une adaptation anisotrope basée sur la hessienne d'une solution numérique (**shock**) avec 1 014 890 points et 2 029 772 mailles et une résolution cible  $n_{\max} = 250\ 000$  pour ce dernier cas. Pour chaque run, une seule adaptation est effectuée sur quatre itérés.

Aucune gradation n'est effectuée. Enfin les points sont initialement réordonnés afin d'obtenir un placement mémoire initial optimal, mais aucune rénumérotation n'est effectuée en cours de calcul.

<sup>51</sup>Le nombre cible de points  $n_{\max}$  est donc à 90% de 500K points.

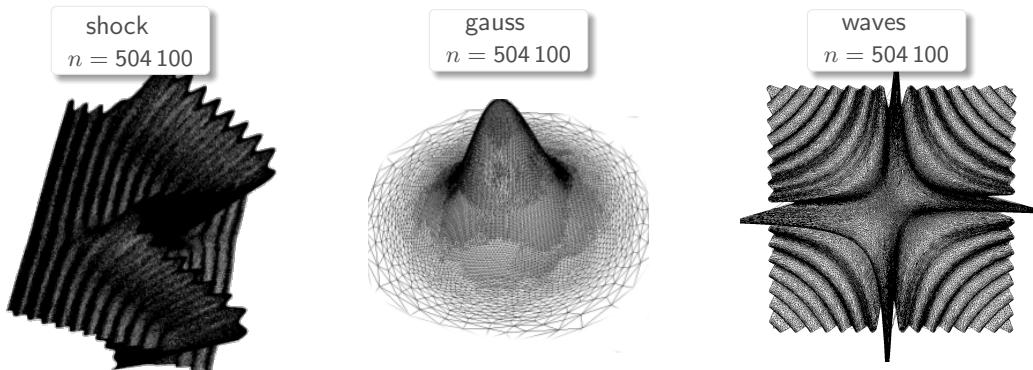


Figure 4.15: Champs de solutions planaires utilisés.

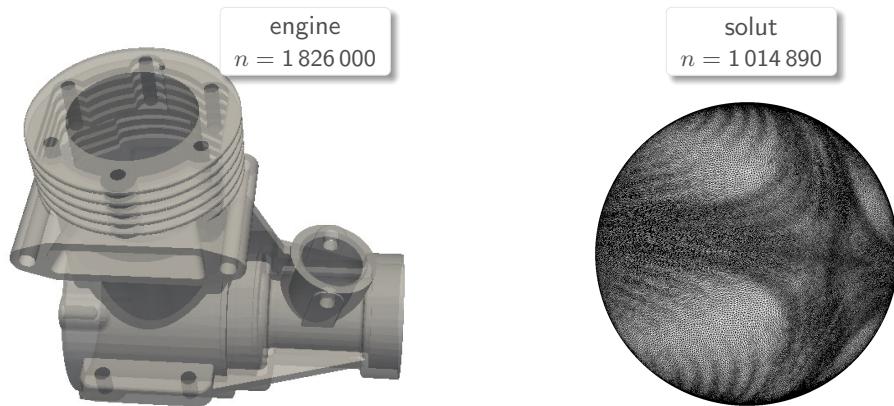


Figure 4.16: Cas-tests surfaciques utilisés.

### 4.3.2 Surcoûts et efficience de nos briques

Ici le but est de montrer rapidement le fait que nos briques pour l'extraction de stable de points et de couplage de mailles, ainsi que le schéma de synchronisation pour la mise à jour du graphe d'incidence soient adaptés à notre contexte. Pour cela nous reportons leur scaling ainsi que les tâches extraites sur les itérés de simplification et de relaxation sur un cas-test planaire (**shock**) et sur **HSW**. Enfin nous donnons une comparaison du temps d'exécution et de surcoût induit par notre schéma de synchronisation avec celle de ROKOS, avec le même cas-test mais sur **HSW** et **KNL** cette fois.

#### 4.3.2.1 Extraction de tâches

SIMPLIFICATION. Pour évaluer notre brique d'extraction d'un stable  $\mathcal{U}$  de points, nous avons profilé l'évolution du surcoût induit comparé au temps de restitution du noyau de simplification sur **HSW** pour le cas-test **shock** à la figure 4.17. Nous avons également comparé le ratio de tâches extraites comparé à une méthode MONTE-CARLO [182] au fur et à mesure des itérés de simplification. Rappelons que notre graphe  $G$  est induit par le graphe primal de la triangulation.

En (1), nous pouvons observer que le surcoût induit par notre heuristique est négligeable devant le temps de restitution et évolue quasi-linéairement en celui-ci<sup>52</sup>. En (2), on voit que le ratio de tâches extraites sur  $|V|$  est un peu mieux comparé à l'approche de LUBY [182]. En fait nous avons remarqué que le nombre de composantes fortement connexes  $\sigma_G$  de  $G$  croît de manière significative car  $G$  devient de plus en plus épars à chaque itéré. Comme nous assignons la plus petite couleur disponible à chaque fois, alors le ratio en question croît proportionnellement à  $\sigma_G$ . Ces résultats confirment que notre heuristique est bien adaptée à notre contexte (figure 4.17).

<sup>52</sup>En pratique, il converge au bout de 3-5 itérés et bon nombre de tâches sont extraites dès le premier itéré.

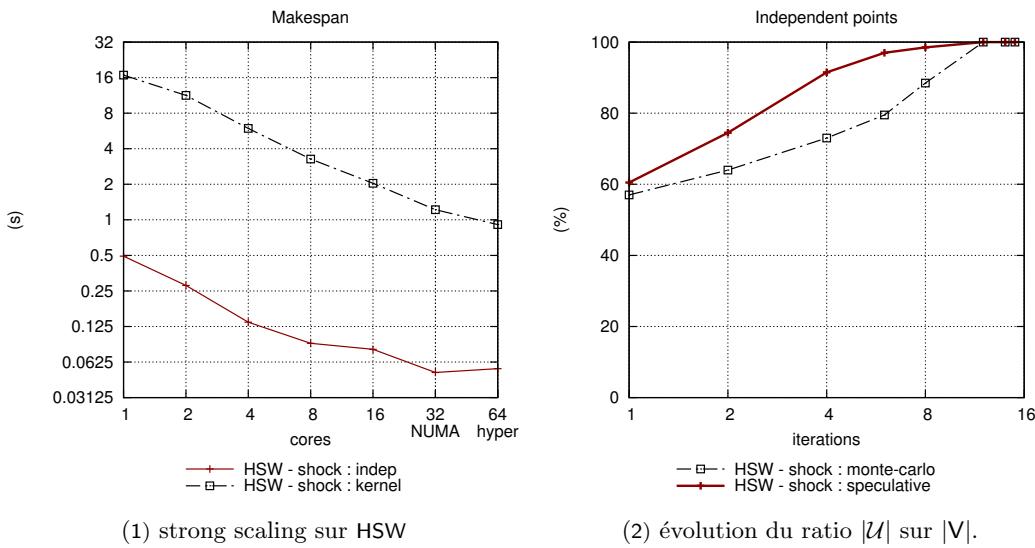


Figure 4.17: Scaling et ratio de tâches extraites sur les itérés de simplification.

RELAXATION. Le scaling de la brique de couplage de mailles ainsi que l'évolution du ratio de tâches extraites sur les itérés de relaxation sont donnés à la figure 4.18. Ici  $G$  est induit par le graphe dual de la triangulation avec initialement un million de mailles. En (1), on voit que le surcoût induit ne représente pas plus de 20% du temps de restitution du noyau, et évolue linéairement à celui-ci. En (2), on voit que le ratio de tâches extraites sur  $|V|$  est sensiblement identique à la version séquentielle et avoisine les 96%, en raison de la planarité et du faible degré du graphe (avec  $\Delta = 3$ ). Notons toutefois que  $|V|$  décroît de manière exponentielle au fur et à mesure des itérés de relaxation : il est donc normal d'observer un écart-type de plus en plus marqué dans les derniers itérés.

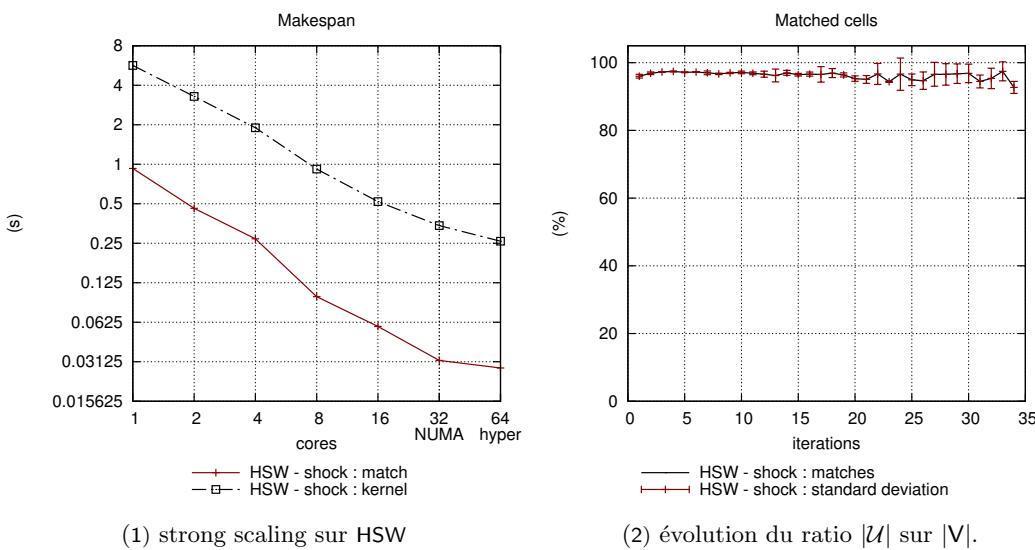


Figure 4.18: Scaling et ratio de mailles couplées sur les itérés de relaxation.

À vrai dire, l'heuristique se termine toujours puisqu'à un moment donné tous les sommets du graphe auront été visités au moins une fois. Bien que la profondeur de recherche soit inégale d'un sommet source à un autre (et donc le nombre d'itérés par sommet source), celle-ci décroît proportionnellement au nombre  $p$  de threads. En effet, plus il y a de threads, moins la recherche est profonde pour chaque source  $s$  : ainsi le déséquilibre de charges tend à décroître de manière significative quand  $p$  tend vers  $n$ .

Notons également que les primitives de synchronisation lock-free sont peu coûteuses et adaptées aux machines récentes (de l'ordre de 15–30 cycles<sup>53</sup> sur une architecture x86 si la variable est dans le cache L1 [176], ce qui est le cas car celle-ci est préchargée avant réécriture).

#### 4.3.2.2 Notre synchronisation

COMPARAISON. Afin d'évaluer l'efficacité de notre schéma de synchronisation, nous nous sommes comparé l'approche de ROKOS. Pour cela, nous avons intégré le schéma de mise à jour différée de Pragmatic [199, 167] dans notre code `trigen`, en nous basant sur son implémentation disponible sur <https://code.launchpad.net/pragmatic>. Nous avons ensuite profilé l'exécution des noyaux sur HSW et KNL. Afin de réduire les effets NUMA, la politique du first-touch a été appliquée et aucune réalllocation des listes d'incidences n'est autorisée dans les deux cas.

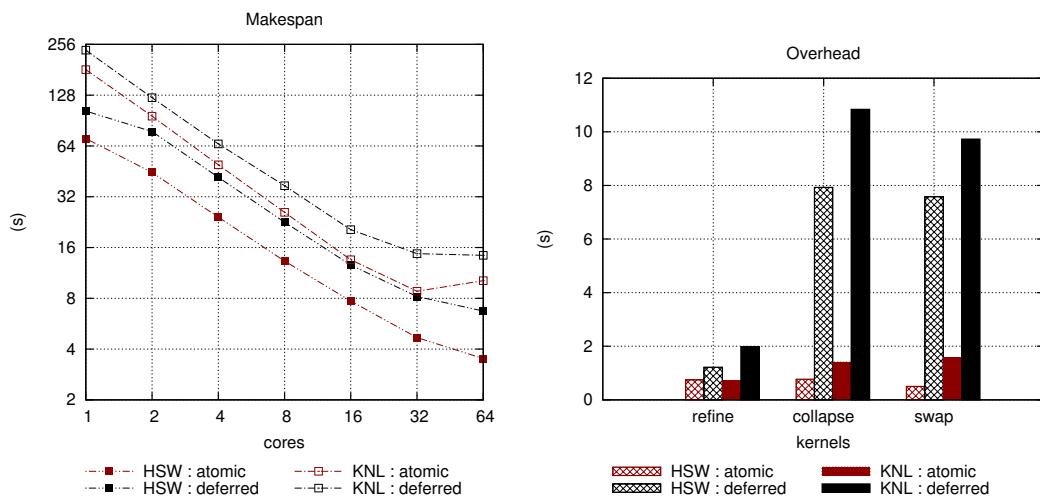


Figure 4.19: Comparaison de performances entre notre synchronisation et celle de Rokos [199, 167].

Le temps de restitution ainsi que sur le surcoût relatif à la synchronisation topologique sont donnés à la figure 4.19. Un premier constat est que le scaling du remailleur est identique dans les deux cas. Par contre le temps de restitution a presque doublé quand nous utilisons la synchronisation différée de ROKOS. En l'occurrence, le temps de restitution est sévèrement impacté par le volume de transferts de données vers et depuis les listes partielles décrites à l'équation 4.1 (page 115). En effet son surcoût est cinq fois plus important que le nôtre pour la simplification et la relaxation.

#### 4.3.3 Scaling et surcoûts

STRONG SCALING. Le temps de restitution moyen des trois cas-tests 2D et des deux cas-tests surficiques sont donnés à la figure 4.20 quand tous les noyaux sont combinés. Rappelons juste que l'hyperthreading est systématiquement utilisé sur KNL (4 threads par core) afin d'atténuer la latence relativement élevée des accès-mémoire. Pour les cas-tests 2D, nous utilisons en priorité la MCDRAM en le spécifiant au préalable au moment de l'exécution par le biais de l'outil Linux `numactl`. Néanmoins cela n'a pas été possible pour les cas-tests surfaciques car l'empreinte mémoire excède les limites de 16 Go de la MCDRAM.

Les cinq cas-tests ont à peu près le même profil de performances, et on obtient un bon strong scaling sur les deux architectures. Conformément à nos attentes, l'exécution est nettement plus lente (deux fois en planaire et quatre fois en surfacique) sur le nœud manycore (KNL) que sur les noeuds multicore (HSW, SKL). En fait cela est normal car leurs cores sont cadencés presque deux fois moins vite d'une part, et que chaque core dispose d'une mémoire beaucoup plus limitée d'autre part.

<sup>53</sup>Cela correspond à peu près au coût d'une division entière ou d'un appel de fonction en C en cycles CPU, cf. <http://ithare.com/infographics-operation-costs-in-cpu-clock-cycles>

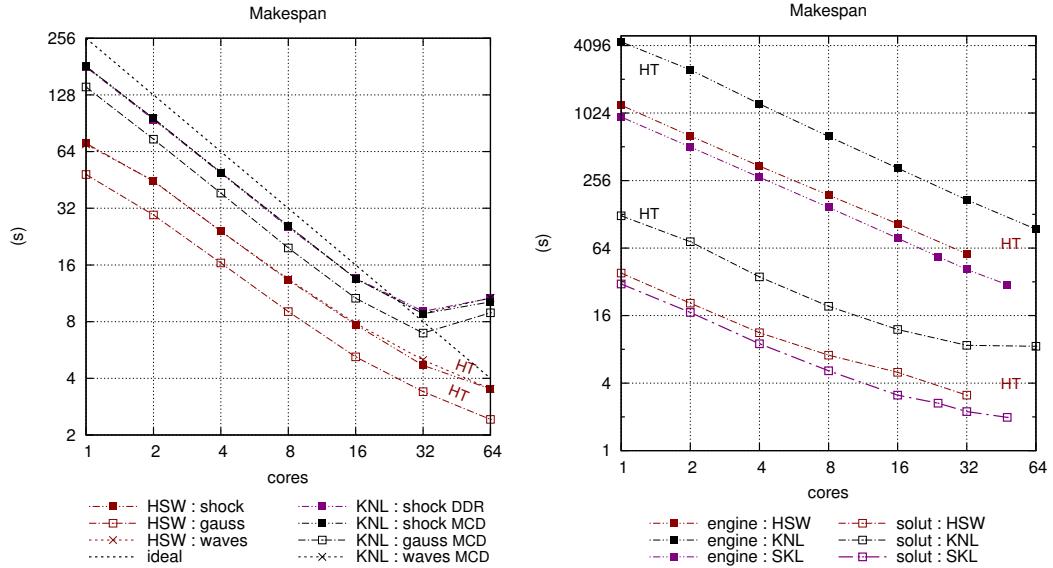


Figure 4.20: Temps de restitution quand tous les noyaux sont combinés.

- **planaire.** En fait nous n'avons obtenu aucune amélioration significative par le recours à la MCDRAM avec ou sans vectorisation. En effet les noyaux du remailleur ne sont pas limités par la bande passante. Ici l'efficacité tombe à 30% sur KNL sur 256 threads (64 cores) en raison de plus fortes contentions à l'accès aux données en cache/mémoire, accentuées par le recours à l'hyperthreading. Néanmoins ils scalent mieux que sur HSW à nombre de cores moins élevé.
- **surfacique.** Comme prévu le temps de calcul est beaucoup plus important qu'en 2D car les noyaux sont clairement plus compliqués et nécessitent plus de calcul. Par contre ils passent mieux à l'échelle du fait qu'ils sont plus compute-intensif. Pour engine, on a une efficacité de 75% sur KNL et 68 % sur HSW-SKL à nombre maximal de cores. Par contre elle est réduite d'un facteur trois pour solut (qui consiste en une simplification) car les cores n'ont pas assez de tâches à traiter. Enfin, notons que engine est plus coûteux que solut en raison des traitements spécifiques des ridges d'une part, et qu'on maintient la même résolution  $n_{\max}$  sur les itérés d'autre part.

Notons enfin qu'on continue d'obtenir une accélération substantielle sur HSW et SKL à nombre maximal de cores quand on active l'hyperthreading. En fait les effets NUMA sont significativement atténus, grâce à notre restructuration locality-aware des accès-mémoire.

**SURCOUTS PAR NOYAU.** La répartition du temps passé sur chaque vague synchrone est donnée à la figure 4.21. Ici les surcoûts induits par la parallelisation sont mis en rouge. Dans notre cas, ils n'excèdent pas 15% du temps de restitution tous noyaux confondus. De plus, ils sont négligeables dans le cas de la contraction et du lissage. En outre, on peut remarquer que ces ratios restent constants en dépit du nombre de threads, et passent à l'échelle dans les mêmes proportions que les autres vagues. En fait, pour

- le raffinement, les opérations sont structurées de sorte qu'aucune extraction explicite de tâches ne soit requise. De plus les requêtes de voisinage induites par la vague de filtrage ne requièrent pas d'avoir des données d'incidence complètement consistantes. En fait les listes d'incidences peuvent contenir des références obsolètes mais toute nouvelle maille doit par contre être référencée dans les listes d'incidence de ses sommets. Ainsi, la vague d'épuration induite par la synchronisation en deux temps n'est effectuée qu'à la toute fin de la procédure pour ce noyau.
- la simplification, la vague de filtrage requiert de reconstruire le voisinage de chaque point  $p$ , afin de trouver le bon voisin candidat  $q$  vers lequel  $p$  doit se fusionner (même en séquentiel). Ainsi

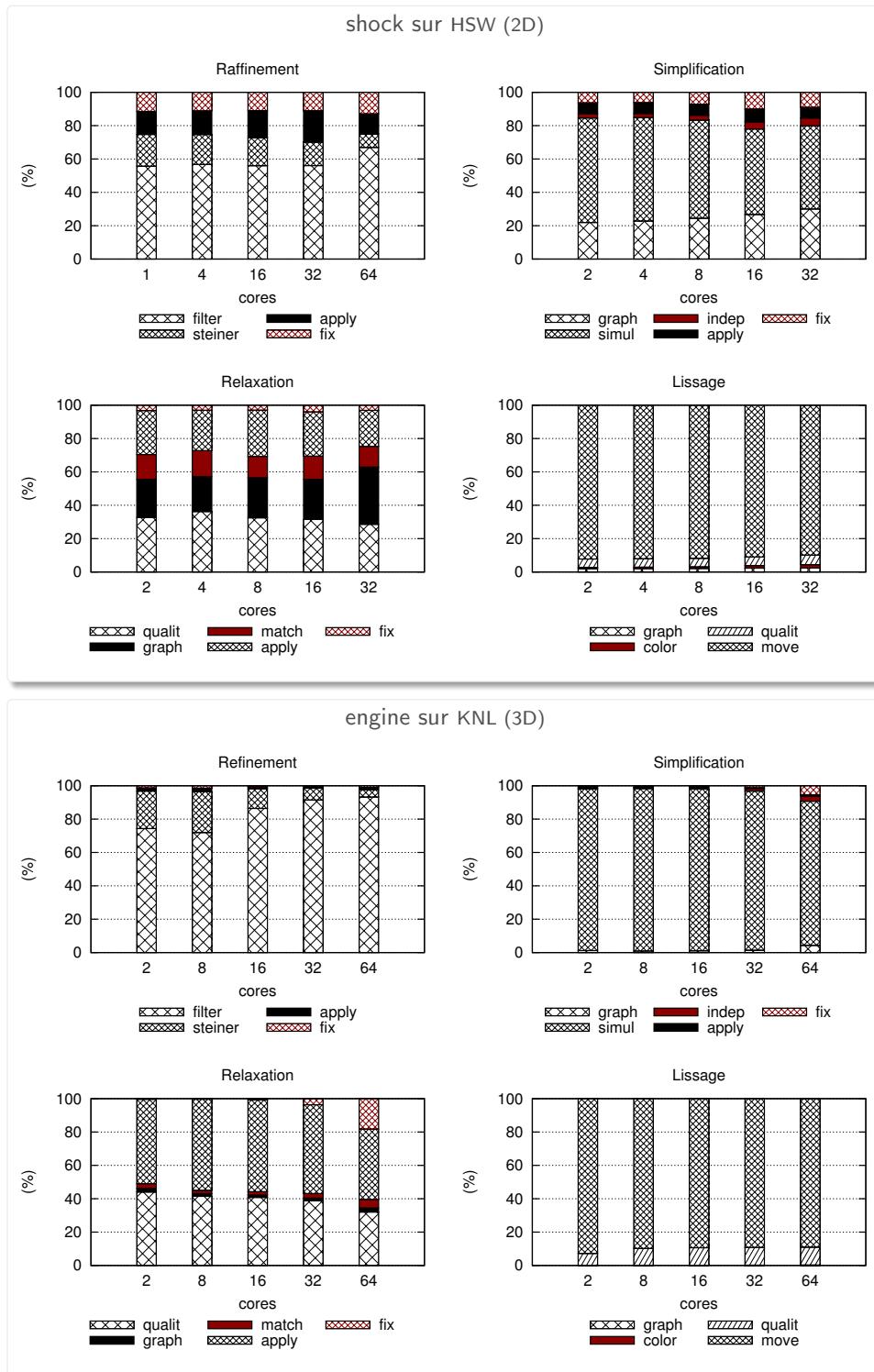


Figure 4.21: Répartition du coût de chaque vague synchrone par noyau.

le graphe primal doit être reconstruit en début d’itéré. Elle consiste essentiellement à des accès-mémoire mais représente près de 22% du temps d’exécution du noyau. Notons que reconstruire le voisinage d’un point n’implique qu’un seul niveau d’indirection dans notre cas, contrairement à une carte combinatoire (listings B.2 et B.1, page 153) : cela permet d’avoir un meilleur scaling.

- la **relaxation de degrés**, le surcout principal provient de la vague de couplage des mailles avec un ratio moyen de 15 %. Ici sa convergence est linéaire en la profondeur de recherche  $\delta_G$  lors de l’extraction des chemins alternants. Notons que cette étape est très irrégulière; et est asymptotiquement en  $O(\log n)$ ,  $n$  étant le nombre de mailles actives. En pratique cette profondeur vaut approximativement 4 avec un ordonnancement statique, et en moyenne la convergence est atteinte au bout de 12 itérés.
- le **lissage**, le graphe primal est construite au tout début du noyau, et aucune vague de synchronisation n’est requise car la topologie reste inchangée. En fait l’unique surcoût est relatif à l’étape de coloration du graphe pour ce noyau. En pratique, un faible nombre d’itérés est requis pour converger (approximativement 3-4).

#### 4.3.4 Débits de tâches par noyau

TASK RATE. Les débits de tâches par noyau sont donnés à la figure 4.22. On peut observer que le raffinement et la relaxation ont de meilleurs débits en termes de débits puisqu’ils sont plus locaux et moins **compute-intensive** que les deux autres. On obtient un débit quasi-linéaire pour chaque noyau à l’exception du raffinement : aucun graphe n’est requis pour celui-ci néanmoins il nécessite plus de synchronisations comparé aux trois autres. En fait le raffinement implique le voisinage de chaque maille, puisque quand une arête est scindée alors les deux mailles incidentes sont également découpées. Néanmoins cette dissection est purement locale car la référence du point de STEINER a déjà été résolue dans une vague à part (page 112). Ainsi les mailles peuvent être traitées individuellement, néanmoins la résolution des références des points requiert plus de synchronisations. Sinon Quand à la relaxation, elle implique la coquille de chaque arête<sup>54</sup> à basculer et donc un voisinage impacté de taille fixe et minimal. A contrario, les noyaux de simplification et de lissage impliquent le voisinage de taille variable et fortement dépendante de l’anisotropie.

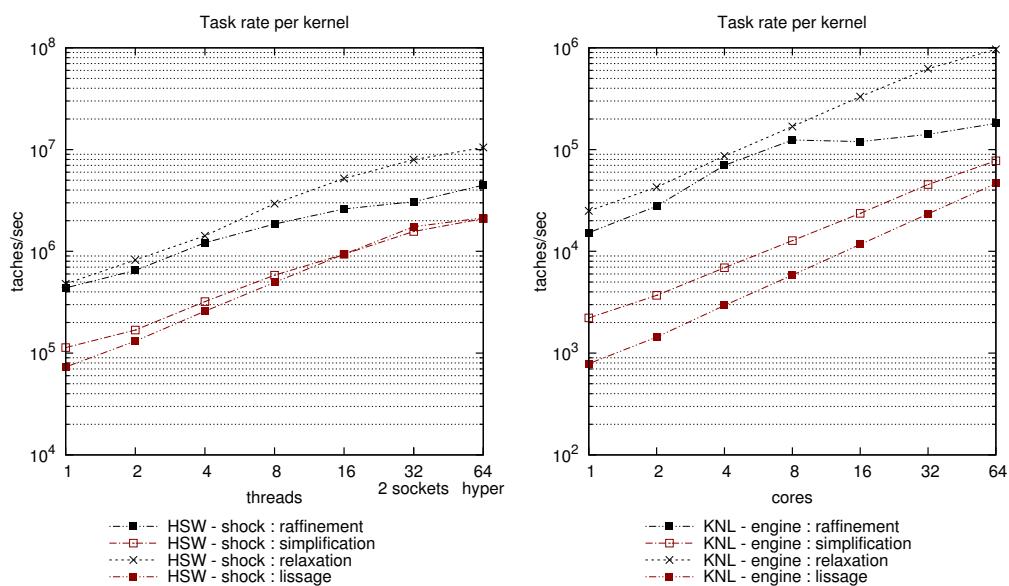


Figure 4.22: Débit de tâches pour chaque noyau.

<sup>54</sup>c'est-à-dire les deux mailles qui lui sont incidentes

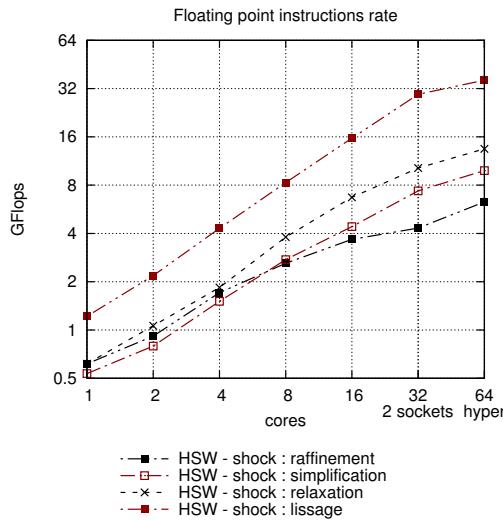


Figure 4.23: Taux d'instructions flottantes par seconde (2D).

**FLOPS.** Le taux d'instructions flottantes traitées par seconde (FLOPS) pour les cas-tests planaires sont données à la figure 4.23. Pour des raisons de limites mémoire, nous n'avons malheureusement pas pu profiler les cas-tests surfaciques<sup>55</sup>. On peut observer que tous les noyaux passent globalement bien à l'échelle. En 2D, la plupart des instructions flottantes sont traitées durant la phase de filtrage, à l'exception du lissage. Pour le raffinement et la simplification, elle implique le calcul de distances géodésiques. Pour la relaxation et le lissage par contre, c'est l'étape où les qualités des mailles sont calculées et mis en cache dans la structure de données du maillage.

Dans notre cas, l'intensité arithmétique<sup>56</sup> reste constant relativement au nombre de cores. Ainsi les transferts de données impliqués par notre schéma de synchronisation n'a pas d'impact significatif sur le taux de FLOPS<sup>57</sup>, même à nombre de threads élevé. Enfin le lissage passe beaucoup mieux à l'échelle que les trois autres noyaux car il a une intensité arithmétique plus élevée. En fait lorsqu'un point est déplacé, sa position ainsi que son tenseur métrique sont interpolés à partir de ses voisins, puis ils sont réajustés itérativement<sup>58</sup> de sorte à rester à l'intérieur de l'enveloppe convexe de son voisinage discret. Ainsi il implique une meilleure réutilisation de données déjà en cache.

## 4.4 CONCLUSION

En résumé, nous avons proposé une approche de parallélisation des noyaux adaptatifs en vue de leur portages sur les architectures multicore et manycore. Elle tient compte des contraintes induites par l'aspect data-driven et data-intensive des noyaux, ainsi que celles du hardware<sup>59</sup>. Elle est adapté aux planaire et surfacique, et toutes ses briques internes sont lock-free. Elle repose sur trois aspects :

- une extraction du parallélisme amorphe pour chaque noyau. Afin d'alimenter suffisamment les cores, nous ne considérons que les conflits liés à la conformité de la topologie, ce qui maximise le nombre de tâches extraites. Pour cela les tâches et les voisinages impactés sont formulées par un graphe propre à chaque noyau, à l'exception du raffinement qui est entièrement local.
- une approche de restructuration des accès-mémoire irréguliers pour atténuer la latence. Pour cela,

<sup>55</sup>Les données intermédiaires générées par le profiler implique une empreinte mémoire énorme.

<sup>56</sup>L'intensité arithmétique est le ratio du nombre d'instructions flottantes sur le nombre d'accès-mémoire

<sup>57</sup>du moins sur les machines avec des capacités de caches normales.

<sup>58</sup>jusqu'à un certain seuil

<sup>59</sup>faible fréquence et mémoire par core, puis coûts élevés et inégaux des accès-mémoire

- les noyaux sont structurés en vagues synchrones afin de réduire la fréquence d'échanges de données et de coalescer les accès en mémoire partagée. Elle concilie les contraintes liés à la localité mémoire<sup>60</sup> et générericité pour la portabilité des performances.
- les emplacements des mailles sont résolues en amont au moment de leur création afin d'éviter les recopies de données locales et pour préserver au mieux la proximité de mailles voisines.
- les réductions de données se font de manière asynchrone<sup>61</sup> ou NUMA-aware<sup>62</sup> selon le cas.
- une **synchronisation à grain fin** pour les mises à jour du graphe d'incidence. Elle est effectivement peu coûteuse, et minimise les transferts et copies de données comparé au schéma de ROKOS.

La solution proposée a été implémentée en C++11 et OpenMP4 sous forme d'une bibliothèque baptisée **trigen**. Pour montrer l'efficacité, nous avons effectué une campagne de benchmarks sur des cas-tests planaires et surfaciques sur deux machines multicore dual-socket (HSW, SKL) et une machine manycore dual-memory (KNL) (page 118). Le scaling obtenu et les ratios de tâches extraites ont montré que nos briques dédiées à l'extraction de stables de points ainsi qu'au couplage de mailles sont réellement adaptées à notre contexte (page 121). Un bon scaling des noyaux est obtenu à nombre de cores élevé, en dépit de la saturation fréquente des caches sur le noeud manycore, ou des effets NUMA sur les noeuds multicore (page 123). De plus les surcoûts induits par la parallélisation restent faible, voire négligeable dans certains cas : ils n'excèdent pas 15% du temps de restitution tous noyaux confondus (page 124). Ces constats sont confortés par les débits de tâches traitées par seconde pour chaque noyau (page 125) ainsi que l'évolution des FLOPS pour les cas-tests planaires (page 126).

\* \* \*

---

<sup>60</sup>saturation fréquente des caches de capacité limitées, latence importante des accès en RAM

<sup>61</sup>pour la mise à jour des listes d'incidence par exemple.

<sup>62</sup>pour la récupération des points ou mailles actives lors du filtrage par exemple.

## **CONCLUSION**

Le but de cette thèse fut de présenter une manière de concevoir des algorithmes numériques irréguliers spécifiquement adaptée aux accélérateurs ou processeurs manycore sur un cas concret. La particularité de cette approche, c'est qu'elle ne repose pas que sur une parallélisation : le choix et la construction même des noyaux numériques est guidé par les contraintes hardware. D'où le terme "co-design" dans l'intitulé. À cette fin nous avons pris les noyaux d'adaptation anisotrope de maillages impliqués dans les simulations en mécanique des fluides numérique comme cas d'étude.

### **Approche et contributions**

EN BREF. En fait on vise à accélérer la phase d'adaptation de la boucle numérique adaptative décrite à la page 11 sur des architectures multicore et manycore. Nous avons vu cela soulevait deux types de contraintes relatives :

- à l'irrégularité des noyaux : ils sont caractérisés par un parallélisme amorphe due aux conflits de tâches dynamiques et non prévisibles d'une part, et une faible réutilisation des données d'autre part. En fait il n'est pas trivial d'exposer un parallélisme constant et maximal à tout instant de leur exécution car le nombre de tâches varie dynamiquement et le fait de traiter une tâche peut invalider d'autres déjà prévues. De plus leurs instructions sont dominées par des accès-mémoire avec une tendance à une saturation fréquente des caches. Ainsi ils ne passent pas aisément à l'échelle surtout sous contrainte mémoire.
- au hardware : les machines manycore intègrent un nombre conséquent de cores avec un très faible rendement (GHZ, GFLOPS, GB) par core; tandis que les machines multicore disposent d'une hiérarchie mémoire<sup>63</sup> à profonde impliquant une latence d'accès de données inégales (NUMA). Pour passer à l'échelle, il faudrait exposer un parallélisme très fin et très élevé pour tirer profit du nombre conséquent de cores faiblement cadencés d'une part, et exposer une forte localité et un fort taux de réutilisation de données pour maximiser le rendement des caches.

Ainsi la réelle difficulté était de concilier ces deux contraintes antagonistes. Pour cela, nous l'avons abordée par une approche de CO-DESIGN numérique-parallèle : nous avons d'abord conçu des noyaux adaptatifs sensibles à la localité mémoire d'une part, puis nous avons proposé une parallélisation tenant compte de leurs aspects **data-driven** et **data-intensive** d'autre part.

DESIGN. En fait cette contrainte de localité est loin d'être triviale puisqu'elle impose de ne recourir qu'à des noyaux très basiques : pas de cavité dynamique, pas de plongement via un atlas, pas de séquence dynamique d'opérations. Pour converger rapidement en erreur et en qualité des mailles néanmoins, il aurait fallu recourir à des noyaux dynamiques tels que ceux dans [120, 123, 97, 135, 136]. Ainsi la vraie difficulté était de proposer des noyaux qui soient aussi efficents que les noyaux de référence tout en impactant un voisinage statique le plus restreint possible. Afin d'exposer une localité maximale, nous n'appliquons que des patterns de modification basiques décrits à la figure 3.2. Pour rester précis et efficents néanmoins, nous nous appuyons sur des briques géométriques avancées. À cette fin, nous avons proposé

- une projection de points basée sur le calcul de géodésiques pour le placement précis des points sur la surface idéale lors de leur création ou déplacement (page 64),
- une relocalisation de points mixte diffusion-optimisation qui améliore la qualité des mailles tout en minimisant la déformation de la surface (page 70).
- un transport optimal de tenseurs métriques qui préserve leurs directions principales lors du déplacement d'un point ou durant la gradation (page 87).

L'efficacité de chaque contribution a été soit formellement prouvée soit évaluée numériquement (page 91). Une partie de ces travaux ont été publiés dans un article [RL18] et présentée oralement à un congrès [LR18].

PORTRAGE. En choisissant avec parcimonie les briques des noyaux, la contrainte de localité fut partiellement résolue. En fait la seconde difficulté a été de trouver une manière de paralléliser les noyaux

---

<sup>63</sup>cache multiples, mémoire locale et éventuellement distante

qui tienne compte de leurs aspects **data-driven** et **data-intensive** ainsi que les contraintes **hardware**. Pour cela nous avons proposée une approche **lock-free** adaptée à la fois aux noyaux planaires et surfaciques et qui s'appuie sur trois aspects :

- une **extraction du parallélisme amorphe** pour chaque noyau basée sur une formulation des tâches et des voisinages impactés en graphes, à l'exception du raffinement qui est entièrement local. Pour un parallélisme maximal, nous ne considérons que les conflits liés à la **conformité** de la topologie (page 105). À cette fin, nous avons conçu deux algorithmes lock-free et massivement multithread pour le calcul de stables et le couplage de mailles (pages 107 et 109); à partir d'une étude expérimentale approfondie des approches récentes [204–206] (annexe B).
- une **approche de restructuration des accès-mémoire irréguliers** pour atténuer la latence. Pour cela, les noyaux sont structurés en vagues synchrones afin de réduire la fréquence d'échanges de données et de coalescer les accès en mémoire partagée (page 111). Afin d'éviter les recopies de données locales et pour préserver au mieux la proximité mémoire de mailles voisines, les emplacements des mailles sont résolus en amont au moment de leur création (page 112). Enfin, les réductions de données se font de manière asynchrone<sup>64</sup> ou NUMA-aware<sup>65</sup> selon le cas (page 113).
- une **synchronisation à grain fin** pour les mises à jour du graphe d'incidence qui minimise les transferts et copies de données comparé à l'état de l'art [199, 167] (page 114).

Pour montrer son efficacité, nous avons mené une campagne de benchmarks sur des cas-tests à la fois planaires et surfaciques sur deux machines dual-socket multicore et une machine dual-memory manycore (page 117). Une partie des résultats obtenus ont été publiés dans trois actes de conférence [RLP16, RLP+17, RL18].

Table 4.4: Publications.

- [RLP16]. Fine-grained locality-aware parallel scheme for anisotropic mesh adaptation.  
Hoby Rakotoarivelo, Franck Ledoux and Franck Pommereau.  
25<sup>th</sup> International Meshing Roundtable, 2016, Washington DC, USA.
- [RLP+17]. Scalable fine-grained metric-based remeshing algorithm for manycore-NUMA architectures. Hoby Rakotoarivelo, Franck Ledoux, Franck Pommereau and Nicolas Le-Goff. Euro-Par: 23<sup>rd</sup> International Conference on Parallel and Distributed Computing, 2017, Santiago de Compostella, Spain. [30% of acceptation]
- [RL18] Accurate manycore-accelerated manifold surface remesh kernels.  
Hoby Rakotoarivelo and Franck Ledoux.  
27<sup>th</sup> International Meshing Roundtable, 2018, Albuquerque NM, USA.
- [LR18] Parallel surface mesh adaptation for manycore architectures.  
Franck Ledoux and Hoby Rakotoarivelo. Communication orale à MeshTrends  
13<sup>th</sup> World Congress in Computational Mechanics, 2018, New-York NY, USA.

## Travaux annexes

Notons qu'une bonne partie du temps de thèse a été dédiée à l'implémentation des algorithmes en vue de leur évaluation expérimentale. Cela a donné lieu au développement d'une bibliothèque baptisée **trigen**, bientôt en open-source. En fait nous avons initialement étudié et implémenté des noyaux anisotropes planaires, avant de les étendre au surfacique. La sortie des résultats a nécessité :

- 7 854 lignes de code en planaire et 12 037 lignes de code en surfacique<sup>66</sup>;
- le déploiement et le profiling sur deux types d'architectures;
- l'implémentation d'algorithmes de graphes [204–206] massivement multithread en C++ vue d'une étude comparative (annexe B);

<sup>64</sup>pour la mise à jour des listes d'incidence par exemple.

<sup>65</sup>pour la récupération des points ou mailles actives lors du filtrage par exemple.

<sup>66</sup>elles sont estimées par l'outil open-source **sloccount** disponible à <https://www.dwheeler.com/sloccount/>.

- l'intégration du schéma de synchronisation de `pragmatic` [167] dans `trigen` pour une comparaison empirique (page 115);
- la génération de cas-tests et fine-tuning sur des instances de matrices `RMAT` [207] et de `CAO`;
- le post-traitment semi-automatisé pour la sortie de résultats par des scripts `Python` et `gnuplot`;
- et le plus fun : le `debugging`.

## Limites et perspectives

Notons que l'approche que nous avons proposée est modulaire : ainsi son efficacité repose sur celles des briques algorithmiques qui la composent. Bien que certaines aient été formellement prouvées<sup>67</sup>, leur design fut guidé par leur efficacité réelle et empirique, en vue de concilier les contraintes induites par l'irrégularité des noyaux avec celles induites par le hardware, tout en étant aussi précis et efficient<sup>68</sup> que les noyaux de référence. En fait les contributions présentées sont des prémisses à plusieurs pistes d'améliorations.

Deux pistes sont prévues à court-terme :

- la finalisation de `trigen` en vue d'une intégration à un framework orienté composants. En fait c'est prévu dans mon travail dans le cadre d'un postdoc au `CMLA`<sup>69</sup> à l'`ENS Cachan`.
- la vérification formelle des briques critiques lock-free notamment les algorithmes suggérés pour l'extraction des tâches ainsi que nos schémas de synchronisations basés sur les primitives atomiques. En fait un piège classique des algorithmes lock-free réside dans la possibilité de `livelock` : ici le but serait de vérifier formellement le caractère `wait-free`<sup>70</sup> de ces briques par le biais d'un `réseau de Petri` par exemple. En fait nous avons déjà initié ce travail, et ce même si aucune référence n'y est faite dans la thèse. Le problème auquel nous nous sommes heurté est qu'entrelacer les étapes de formalisation et d'optimisation des noyaux est difficile dans la durée car les différentes mises à jour remettaient en cause le modèle formel proposé.

À moyen terme, deux pistes sont envisagées :

- une analyse approfondie des briques notamment les noyaux de relaxation et de lissage. Nous avons vu que l'égalisation efficiente des degrés n'est pas un problème trivial en raison des nombreux minima locaux. En fait nous avons suggéré une heuristique gloutonne assez efficiente en pratique mais qui n'est pas garanti de converger. Pour le lissage, la routine d'optimisation implique une étape de projection explicite à chaque itéré. Une piste intéressante serait d'intégrer directement la contrainte de projection dans la formulation du problème à l'aide du `LAGRANGIEN` puis de résoudre le problème d'optimisation sans contrainte à l'aide de l'algorithme d'`UZAWA` : ce serait intéressant de comparer leur efficacité en termes de convergence. Enfin il serait également intéressant d'analyser plus finement les algorithmes dédiés à l'extraction des tâches en vue d'exposer leurs ratios d'approximation et leur complexité parallèle. Cela permet de caractériser théoriquement leur efficience en termes de ratio entre qualité de la solution extraite et surcoût induit.
- une parallélisation à grain multiple dédiée aux clusters de noeuds manycore<sup>71</sup>. Ainsi l'approche de restructuration par vagues synchrones que nous avons initiée peut adaptée afin de tenir compte de la hiérarchie mémoire par le biais du `bridging-model MULTI-BSP` [170]. Il fournit un modèle de coût des vagues synchrones qui pourrait être utilisé pour le `rééquilibrage dynamique` et `NUMA-aware` des charges entre les noeuds de calcul. In fine, la structure de l'algorithme reste inchangée néanmoins chaque brique doit être mise à jour afin de supporter la communication intra-noeud et inter-noeud<sup>72</sup> des processus tout en gardant le même profil de performances. Enfin une comparaison des noyaux avec leur version `task-based` avec `rééquilibrage par vol de tâches`

<sup>67</sup>notamment le transport parallèle de tenseurs métriques ainsi la preuve d'équivalence des seuils de gradation.

<sup>68</sup>en termes de convergence en erreur et qualité de mailles

<sup>69</sup>Centre de mathématiques et leurs applications, UMR 8536.

<sup>70</sup>Cette propriété assure qu'au moins un thread progresse à tout instant de l'exécution de l'algorithme.

<sup>71</sup>ce sera l'architecture type des futurs calculateurs exaflopiques.

<sup>72</sup>mémoire partagée en intra-noeud et mémoire distribuée en inter-noeud

serait intéressant. Compte-tenu de leur irrégularité, cela permettrait de profiler la sensibilité des noyaux quand la localité est privilégié au profit de l'asynchronisme et vice-versa.

À plus long terme, une piste envisageable serait de proposer une **approche unifiée** des problèmes irréguliers. En effet ils n'ont pas le même profil d'irrégularité (dépendances de données très dynamiques versus répartition très inégale des charges) et nécessitent souvent une approche problème-spécifique (voir [164] pour une synthèse). Pour cela une piste possible serait de définir des **classes d'équivalence** en vue de proposer une méthodologie par classe.

\* \* \*

## **Part III**

---

### **Annexe**

# Noyaux numériques

Dans ce chapitre, nous décrivons les noyaux numériques additionnels que nous utilisons pour la reconstruction de la surface, ainsi que pour la construction du champ de tenseurs métriques pour la répartition anisotrope des points sur la surface. Les détails d'implémentation sont également donnés.

A.1	Reconstruction de la surface . . . . .	133
A.1.1	Paramétrisation quartique . . . . .	133
A.1.2	Jacobienne et aire . . . . .	133
A.2	Champ de tenseurs métriques . . . . .	134
A.2.1	Reconstruction de la hessienne . . . . .	134
A.2.2	Reconstruction des courbures . . . . .	137
A.2.3	Couplage des deux champs . . . . .	138
A.2.4	Interpolation des tenseurs . . . . .	139

## A.1 RECONSTRUCTION DE LA SURFACE

### A.1.1 Paramétrisation quartique

Pour chaque maille  $K$ , la région de  $\Gamma$  qu'elle représente est approchée par une surface polynomiale  $\Gamma_K = \sigma(\hat{K})$  où  $\hat{K} = \{(u, v) \in [0, 1]^2, 1 - u - v \geq 0\}$  est l'élément fini de référence de  $\mathbb{R}^2$ . Il représente l'espace paramétrique de  $\mathbb{R}^2$  associé à  $\sigma$ . Pour exprimer les coordonnées d'un point de  $K$  dans  $\hat{K}$ , il suffit de prendre ses coordonnées barycentriques dans  $K$ . Comme  $\Gamma_K$  est une surface quartique, alors chaque composante de  $\sigma : [0, 1]^2 \rightarrow \mathbb{R}^3$  implique un polynôme de degré 4. Dans notre cas, il s'agit de polynômes quartiques de Béziers, tel que tout point de  $K$  de coordonnées barycentriques  $(u, v)$  est interpolé comme suit :

$$\sigma(u, v) = \sum_{\substack{0 \leq i, j, k \leq 4 \\ i+j+k=4}} \mathbf{B}_{ijk}^4(u, v) b_{i,j,k} = \sum_{i=0}^4 \sum_{j=0}^{3-i} \frac{4!}{i!j!k!} u^i v^j w^k b_{i,j,k}, \text{ avec } \begin{cases} k = 3 - j \\ w = 1 - u - v \end{cases} \quad (\text{A.1})$$

où les  $b_{i,j,k} \in \mathbb{R}^3$  sont les 15 points de contrôle illustrés à la figure 3.7 mais qu'il reste à spécifier. En particulier, tout point d'une courbe frontière  $\gamma_0, \gamma_1$  et  $\gamma_2$  du triangle quartique  $\Gamma_K$  s'exprime par :

$$\begin{aligned} \forall t \in [0, 1], \quad & \gamma_0(t) = \sigma(1-t, t) \\ & \gamma_1(t) = \sigma(0, t) \\ & \gamma_2(t) = \sigma(t, 0) \end{aligned} \quad (\text{A.2})$$

### A.1.2 Jacobienne et aire

JACOBIENNE. Il nous est souvent utile de calculer la matrice jacobienne  $J_\sigma \in \mathbb{R}^{3 \times 2}$  de la paramétrisation  $\sigma$ . Elle permet d'estimer le gradient local de  $\Gamma$  en tout point  $p = \sigma(u, v) = (x, y, z)$  de  $K$ . Elle fournit aussi une base orthonormale du plan tangent de  $p$ , ce qui permet de trouver le vecteur normal en ce point. De manière générale, une jacobienne permet de passer d'un système de coordonnées locales à un autre (repère de Frenet vers base canonique de  $\mathbb{R}^3$  par exemple). Dans notre cas, on a :

$$\begin{aligned} J_\sigma(u, v) &= (\partial_u p, \partial_v p) = \nabla \Gamma_K(p) \\ \text{avec } & \left( \begin{matrix} \partial_u p \\ \partial_v p \end{matrix} \right) = 4 \left( \begin{matrix} \sum_{i,j,k} \mathbf{B}_{ijk}^3(u, v) (b_{i+1,j,k} - b_{i,j,k+1}) \\ \sum_{i,j,k} \mathbf{B}_{ijk}^3(u, v) (b_{i,j+1,k} - b_{i,j,k+1}) \end{matrix} \right), \text{ où } \begin{cases} 0 \leq i, j, k \leq 3, \text{ et} \\ i + j + k = 3 \end{cases} \end{aligned} \quad (\text{A.3})$$

Ainsi la normale sur  $p$  s'obtient tout simplement par le produit vectoriel des composantes de  $J_\sigma(\hat{p})$  :

$$\mathbf{n}(p) = \frac{\partial_u p \times \partial_v p}{\|\partial_u p \times \partial_v p\|} \quad (\text{A.4})$$

Munie de cette paramétrisation locale, on peut reconstruire intégralement la variété  $\Gamma$  sur  $\mathcal{T}_h$ . En connectant les triangles quartiques, on peut désormais (1) projeter tout point  $p$  nouvellement créé ou modifié sur une région de  $\Gamma$ , (2) calculer son vecteur normal  $\mathbf{n}(p)$ , (3) estimer les quantités relatives à la variation de  $\Gamma$  en ce point (gradient, hessienne, courbure), ce qui permet (4) d'estimer l'erreur d'approximation de  $\Gamma$  dans un voisinage local à  $p$ .

AIRE. L'aire du patch quartique  $\Gamma_K$  est défini par [1](§3.2.2) :

$$|\Gamma_K| = \int_K \det[J_x^\top J_x]^{1/2} dx \approx |\mathcal{K}| \sum_{c_i \in K} \det[J_{c_i}^\top J_{c_i}]^{1/2}. \quad (\text{A.5})$$

Ici les  $c_i$  sont les trois points de quadrature de  $K$  pour l'intégration numérique. Ils correspondent aux milieux des arêtes. Enfin la matrice  $J_x \in \mathbb{R}^{3 \times 2}$  correspond à la jacobienne de la paramétrisation au point  $x$ . Notons juste que le produit  $J_x^\top J_x$  n'est pas anodin car il correspond à la première forme fondamentale de  $\Gamma$  au point  $p \in \Gamma_K$  : il définit le produit scalaire local à  $T_p \Gamma$ .

Ainsi il reste à déterminer les points de contrôle  $(b_{i,j,k})$  pour chaque maille  $K$ , avec  $0 \leq i, j, k \leq 4$ ,  $i + j + k = 4$ . Ils se déduisent des vecteurs normaux aux sommets de  $K$  mais doivent être choisis judicieusement afin de tenir compte des contraintes de continuité aux courbes frontières et aux sommets de  $K$ .

## A.2 CHAMP DE TENSEURS MÉTRIQUES

### A.2.1 Reconstruction de la hessienne

La construction du tenseur de calcul fait intervenir la hessienne  $H_u$  de la solution  $\mathbf{u}$  qui n'est connue qu'aux points de la triangulation. Afin d'avoir une métrique précise, il est nécessaire d'avoir une bonne approximation de  $H_u$ . Il existe dans la littérature plusieurs manières de la reconstruire : (1) par les *moindres carrés*, (2) par la *formule de Green*, ou (3) par la *double L<sup>2</sup>-projection* [103]. Nous avons opté pour cette dernière méthode car elle ne tend pas à lisser le champ calculé contrairement à (1), et est plus simple à calculer que (2) en ayant la même précision. Ainsi on calcule d'abord  $H_{u|K}$  sur chaque maille de  $S$  et on intègre ensuite sur  $S$ . En particulier, nous montrons comment inférer  $H_{u|K} = \nabla^2 \mathbf{u}_{h|K}$  sur la variété  $\partial\Omega$  en absence de données supplémentaires sur  $\mathbf{u}$ .

PRINCIPE. L'idée est d'abord de reconstruire le gradient  $\nabla \mathbf{u}$  en chaque point de la triangulation. Ensuite on ré-applique le même opérateur  $\nabla$  sur l'interpolé de chaque composante du gradient. Au sein d'une maille, la solution discrète  $\mathbf{u}_h$  s'écrit comme une combinaison linéaire des fonctions de base  $(\psi_j)_{j=1}^3$  du  $\mathbb{P}^1$ -élément fini  $K$ . Pour toute maille  $K : (p_0, p_1, p_2)$ ,  $\mathbf{u}$  et son gradient  $\nabla \mathbf{u}$  sont calculés dans  $K$  par :

$$\mathbf{u}_{h|K}(x) = \sum_{j=0}^2 \mathbf{u}(p_j) \psi_j(x) \quad (\text{A.6})$$

$$\nabla \mathbf{u}_{h|K}(x) = \sum_{j=0}^2 \mathbf{u}(p_j) \nabla \psi_j(x). \quad (\text{A.7})$$

Dans le cas planaire, on se ramène au triangle de référence  $\hat{K}$  pour le calcul des gradients des fonctions de base. Ensuite, on construit une transformation affine  $\varsigma : \mathbb{R}^2 \rightarrow \mathbb{R}^2$  permettant d'exprimer les sommets de  $\hat{K}$  vers ceux de  $K$ , avec  $\varsigma(u, v) = [\sum_{j=0}^2 x_j \hat{\lambda}_j(u, v), \sum_{k=0}^2 y_k \hat{\lambda}_j(u, v)] = [x, y]$ . Dans le triangle de référence, les  $(\hat{\lambda}_j)$  correspondent aux coordonnées barycentriques avec  $(\hat{\lambda}_0, \hat{\lambda}_1, \hat{\lambda}_2) = (1 - u - v, u, v)$ . La matrice jacobienne  $J_\varsigma$  associée à cette transformation est :

$$J_\varsigma = \begin{pmatrix} \partial_u x & \partial_v x \\ \partial_u y & \partial_v y \end{pmatrix} = \begin{pmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{pmatrix} \text{ avec } \det(J_\varsigma) = 2|K| \quad (\text{A.8})$$

La transformation affine  $\varsigma$  est inversible si  $p_0, p_1$  et  $p_2$  ne sont pas alignés. De plus, les coordonnées barycentriques sont conservées :  $\hat{\lambda}_j(u, v) = \lambda_j(x, y)$ . Ici les fonctions de base  $\psi_j$  coïncident avec les coordonnées barycentriques  $\lambda_k$ . Dans ce cas, il suffit de calculer  $\nabla \lambda_j$  pour obtenir  $\nabla \psi_j$ .

Par composition des dérivées, on a :

$$\nabla \psi_j = \nabla \lambda_j = \begin{pmatrix} \partial_x \lambda_j \\ \partial_y \lambda_j \end{pmatrix} = \begin{pmatrix} \partial_u \hat{\lambda}_j & \partial_x u + \partial_v \hat{\lambda}_j & \partial_x v \\ \partial_u \hat{\lambda}_j & \partial_y u + \partial_u \hat{\lambda}_j & \partial_y v \end{pmatrix} \quad (\text{A.9a})$$

$$= \begin{pmatrix} \partial_x u & \partial_x v \\ \partial_y u & \partial_y v \end{pmatrix} \begin{pmatrix} \partial_u \hat{\lambda}_j \\ \partial_v \hat{\lambda}_j \end{pmatrix} \quad (\text{A.9b})$$

$$= (J_\varsigma^{-1})^\top \nabla \hat{\lambda}_j \quad (\text{A.9c})$$

$$= J_{\varsigma^{-1}}^\top \varsigma(\nabla \hat{\lambda}_j) \quad (\text{A.9d})$$

EXTENSION EN SURFACIQUE. Ici la situation est moins claire car on ne dispose pas d'un triangle de référence de  $\mathbb{R}^3$ . Dans le cas volumique, le gradient en un point surfacique  $p_i \in \partial\Omega$  est (1) soit calculé à partir des tétraèdres incidents à  $p_i$ , auquel cas  $J_\varsigma \in \mathbb{R}^{3 \times 3}$  est bien définie et inversible, (2) soit fixé par une condition aux limites de type Neumann. En absence de données supplémentaires, l'idée est de se ramener au cas planaire, par une série de transformations. Pour cela, l'idée est (1) de projeter le stencil  $S$  sur le plan tangent du point  $p$ , et d'exprimer chaque  $p_i \in \partial S$  dans le repère local  $(u, v)$  du plan tangent de  $p$ , puis (2) calculer  $\nabla \psi_j$  en coordonnées  $(u, v)$ , et enfin (3) le ramener dans  $\mathbb{R}^3$ . En notant  $P$  la matrice de projection de  $T_p \Gamma$  vers  $\Gamma$  et  $J_\sigma$  la jacobienne de  $\sigma : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ , on a :

$$\nabla \mathbf{u}_h|_K(x) = \sum_{j=0}^2 \mathbf{u}(p_j) P J_\sigma \nabla \psi_j \quad (\text{A.10a})$$

$$= \sum_{j=0}^2 \mathbf{u}(p_j) P J_\sigma (J_\varsigma^{-1})^\top \nabla \hat{\lambda}_j \quad (\text{A.10b})$$

Finalement, le gradient reconstruit est la moyenne pondérée des gradients des éléments appartenant au stencil  $S$ .

$$\nabla_R \mathbf{u}_h = \frac{1}{|S|} \int_S \nabla \mathbf{u}_h \quad (\text{A.11a})$$

$$= \frac{1}{|S|} \sum_{K \in S} \int_K \nabla \mathbf{u}_h \quad (\text{A.11b})$$

$$\approx \frac{\sum_{K \in S} |K| \nabla \mathbf{u}_h|_K}{\sum_{K \in S} |K|} \quad (\text{A.11c})$$

DÉDUCTION DE LA HESSIENNE. En ré-appliquant l'opérateur  $\nabla_R$  sur l'interpolé de chaque terme du gradient, et en symétrisant la matrice résultante  $\nabla_R^2 \mathbf{u}$ , on reconstruit la hessienne  $H_{\mathbf{u}, R}$  comme suit :

$$H_{\mathbf{u}, R} = \frac{1}{2} (\nabla_R^2 \mathbf{u} + \nabla_R^2 \mathbf{u}^\top) \quad (\text{A.12})$$

où pour tout point  $x : \nabla_R^2 \mathbf{u}(x) = \nabla_R (\nabla_R \mathbf{u}(x))^\top$  (A.13)

$$= \frac{1}{\sum_{K \in S} |K|} \left( \begin{array}{c} \sum_{K \in S} |K| \nabla^\top (\Pi_c(\partial_x \mathbf{u})|_K) \\ \sum_{K \in S} |K| \nabla^\top (\Pi_c(\partial_y \mathbf{u})|_K) \\ \sum_{K \in S} |K| \nabla^\top (\Pi_c(\partial_z \mathbf{u})|_K) \end{array} \right) \quad (\text{A.14})$$

$$= \frac{1}{\sum_{K \in S} |K|} \left( \begin{array}{c} \sum_{K \in S} |K| \nabla^\top (\sum_{j=0}^2 \partial_x \mathbf{u}_h(p_j) \nabla \psi_j(x)) \\ \sum_{K \in S} |K| \nabla^\top (\sum_{j=0}^2 \partial_y \mathbf{u}_h(p_j) \nabla \psi_j(x)) \\ \sum_{K \in S} |K| \nabla^\top (\sum_{j=0}^2 \partial_z \mathbf{u}_h(p_j) \nabla \psi_j(x)) \end{array} \right) \quad (\text{A.15})$$

avec  $\Pi_c$  étant l'opérateur d'interpolation de Clément. La procédure complète est résumée à l'algorithme A.1.

**Algorithme A.1: Reconstruction de la hessienne**

**1. Reconstruction des gradients.**

**pour** chaque point  $p$  **faire en parallèle**

calculer une base locale  $J = (\partial_u p, \partial_v p)$  du plan tangent de  $p$ .

$\mathbf{g} = \vec{0}$ .

$\lambda = 0$ .

**pour** chaque maille  $K$  incidente à  $p$  **faire**

calculer  $\tilde{K}$  le projeté  $K$  sur le plan tangent de  $p$ .

exprimer les points de  $\tilde{K}$  dans la base  $J$ .

calculer la jacobienne  $R$  de  $\tilde{K}$  ainsi que  $\omega = \det(R)$ .  $\triangleright$  voir équation (A.8).

$\mathbf{v}_0 = \omega(v_1 - v_2, u_2 - u_1)$ .

$\mathbf{v}_1 = \omega(v_2 - v_0, u_0 - u_2)$ .  $\triangleright$  gradients des fonctions de base  $\psi_i$  de  $\tilde{K}$ .

$\mathbf{v}_2 = \omega(v_0 - v_1, u_1 - u_0)$ .

$\lambda = \lambda + |K|$ , avec  $|K| = \omega/2$ .  $\triangleright$  aire du stencil.

$\mathbf{w} = \vec{0}$ .

**pour** chaque direction  $\mathbf{v}_i$  **faire**

$\mathbf{w} = \mathbf{w} + \mathbf{u}[x_i] \mathbf{v}_i$ .  $\triangleright$  voir équation (A.10).

$\mathbf{g} = \mathbf{g} + |K| \mathbf{w}$ .  $\triangleright \mathbf{g} = \sum_{K \in S} |K| \sum_{i=0}^2 \mathbf{u}[x_i] \mathbf{v}_i$ .

**fin**

stocker  $\nabla \mathbf{u}[p] = \frac{1}{\lambda} J \mathbf{g}$ .  $\triangleright$  voir équation (A.11).

**barrière**

**2. Reconstruction des matrices hessiennes.**

**pour** chaque point  $p$  **faire en parallèle**

calculer une base locale  $J = (\partial_u p, \partial_v p)$  du plan tangent de  $p$ .

$M = 0_{3 \times 3}$ .

$\lambda = 0$ .

**pour** chaque maille  $K$  incidente à  $p$  **faire**

calculer  $\mathbf{v}_i$  et  $\lambda$  comme précédemment.  $\triangleright$  trop coûteux en mémoire.

$H = 0_{3 \times 3}$ .

**pour** chaque direction  $\mathbf{v}_i$  **faire**

$\mathbf{w} = J \mathbf{v}_i$ .  $\triangleright$  gradient des  $\psi_i$  dans la base canonique de  $\mathbb{R}^3$ .

$H_{i,0} = H_{i,0} + \partial_x \mathbf{u}[x_i] \mathbf{w}$ .

$H_{i,1} = H_{i,1} + \partial_y \mathbf{u}[x_i] \mathbf{w}$ .  $\triangleright$  voir équation (A.15).

$H_{i,2} = H_{i,2} + \partial_z \mathbf{u}[x_i] \mathbf{w}$ .

**fin**

$M = M + |K| H$ .  $\triangleright M = \left( \sum_{K \in S} |K| \sum_{i=0}^2 \partial_k \mathbf{u}[x_i] \mathbf{w} \right)_{k \in [x,y,z]}$ .

**fin**

stocker  $H_{\mathbf{u},p} = \frac{1}{2\lambda}(M + M^T)$ .  $\triangleright$  symétrisation, voir équation (A.12).

**barrière**

supprimer les gradients stockés auparavant.

**retourner**  $(H_{\mathbf{u},x})_{x \in \mathcal{T}_h}$ .

### A.2.2 Reconstruction des courbures

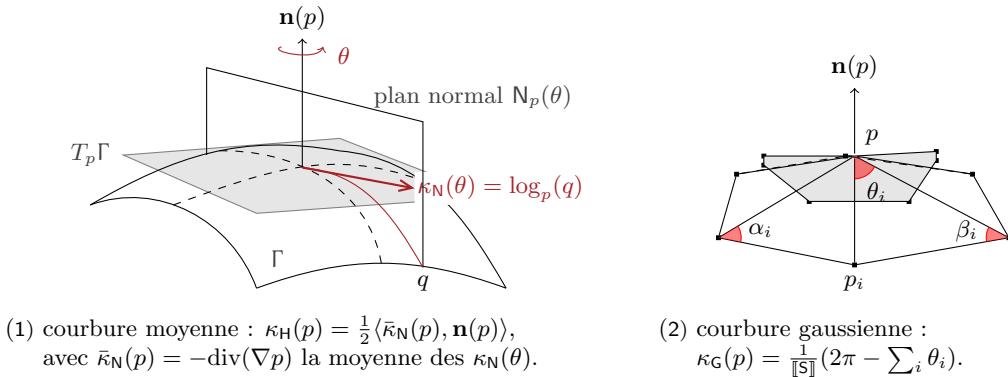


Figure A.1: Reconstruction numérique de la courbure moyenne  $\kappa_H = \frac{1}{2}(\kappa_0 + \kappa_1)$  et gaussienne  $\kappa_G = \kappa_0 \kappa_1$  d'un point de la variété. Notons que  $\kappa_G$  est intrinsèque à la variété et ne dépend pas de son plongement, tandis que  $\kappa_H$  est extrinsèque et varie selon le référentiel.

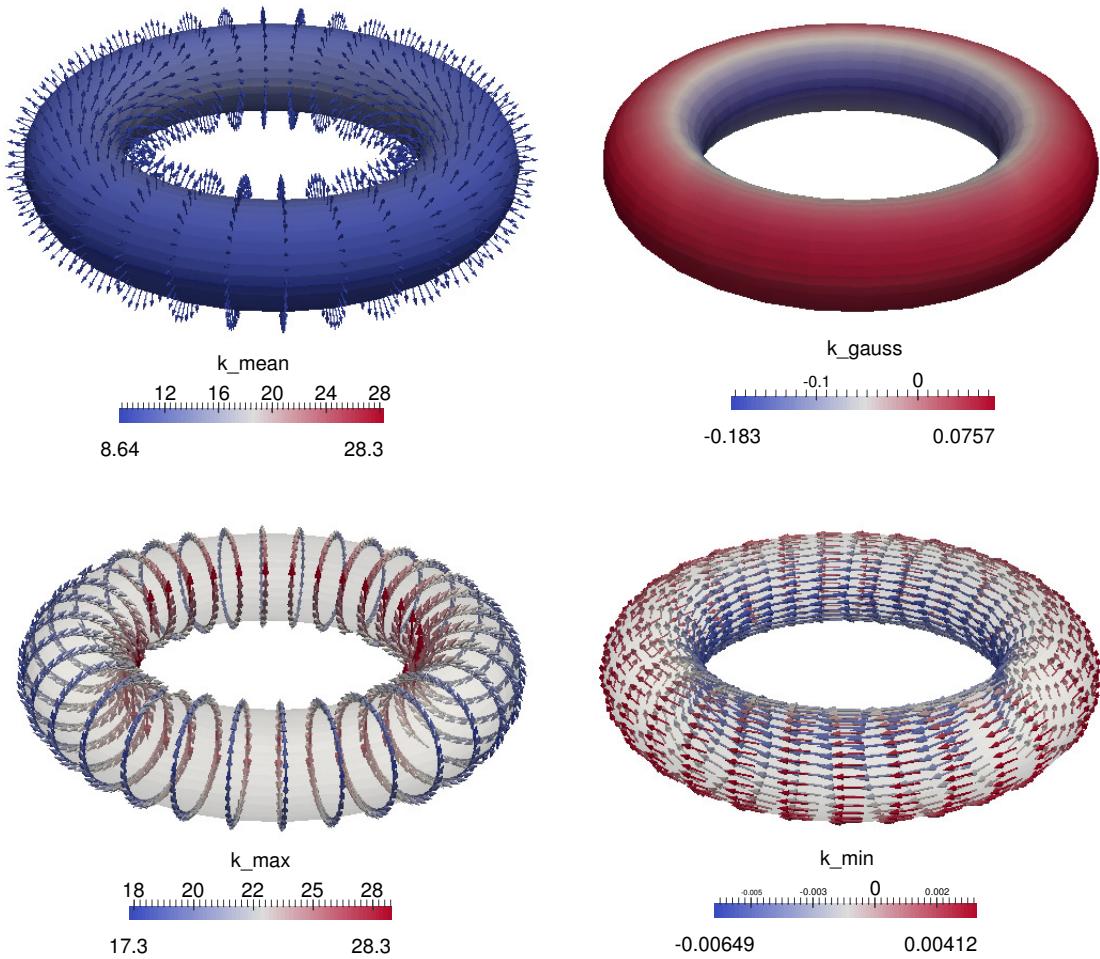


Figure A.2: Reconstruction numérique des composantes du tenseur de courbure sur un tore. Les courbures principales  $\kappa_i$  correspondent aux valeurs propres associées aux directions tangentielle  $\mathbf{v}_i$ . Dans notre cas, les  $\kappa_i$  sont reconstruits à partir des courbures moyennes  $\kappa_H$  et gaussiennes  $\kappa_G$  avec  $(\kappa_0, \kappa_1) = (\kappa_H + \sigma, \kappa_H - \sigma)$  où  $\sigma = \sqrt{\kappa_H^2 - \kappa_G}$ .

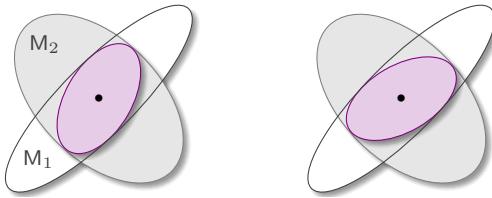


Figure A.3: Intersections possibles de deux tenseurs. Le tenseur résultant correspond au plus grand ellipsoïde contenu dans l'intersection des boules géodésiques de  $x$  associés à  $M_1$  et  $M_2$ . Notons que son orientation dépend du fait qu'on applique  $M_1 \cap M_2$  ou  $M_2 \cap M_1$ .

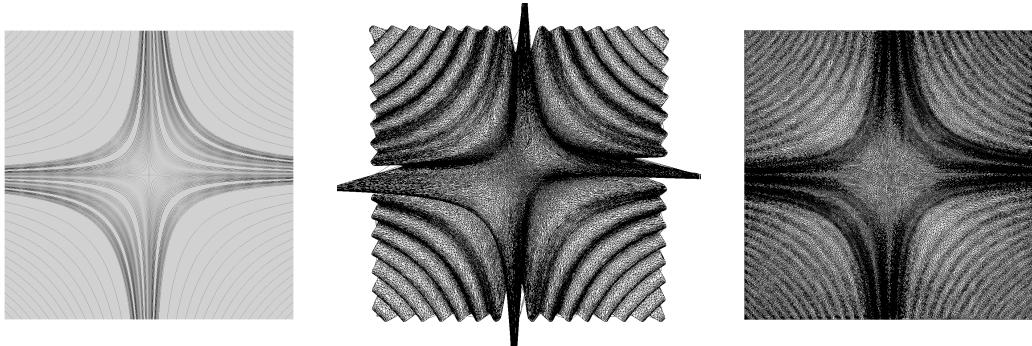


Figure A.4: Triangulation adaptée à l'erreur d'une solution numérique  $\mathbf{u}$  en 2D. À gauche, on a la répartition souhaitée des points sur un domaine carré  $\Omega = [0, 1]^2$  discréteisé en une grille uniforme. Au milieu, on a la triangulation uniforme de la variété riemannienne induite par les hessiennes locales de  $\mathbf{u}$  sur  $\Omega$ . À droite, la triangulation adaptée obtenue en supprimant les coordonnées en  $z$ , avec  $n = 10^5$  points.

### A.2.3 Couplage des deux champs

À ce stade nous avons défini deux métriques distinctes : l'une relative au tenseur de courbure et l'autre à la hessienne de  $\mathbf{u}$ . Cependant nous voulons une métrique unique qui contrôle à la fois l'approximation de la variété et l'erreur de  $\mathbf{u}$  sur la triangulation. Pour cela, nous recourons à l'intersection de tenseurs. Notons que c'est une procédure classique dans la littérature [59]. Nous la rappelons juste de manière simplifiée pour la cohérence du chapitre et parce qu'elle est aussi utilisée pour la gradation.

**INTERSECTION DE TENSEURS.** Notons indistinctement  $M_1$  et  $M_2$  les deux tenseurs associés à un point  $x$  de  $\mathcal{T}_h$ , et exprimés dans une base locale au plan tangent de  $x$ . Ici on cherche à garder la contrainte de taille la plus restrictive dans toutes les directions imposées par  $M_1$  et  $M_2$ . Intuitivement, l'idée est de considérer le plus grand ellipsoïde inclus dans l'intersection des boules géodésiques unités associés  $M_1$  et  $M_2$  comme illustré sur la figure A.3.

En fait, on cherche à les exprimer dans une base commune  $P$  tel qu'ils soient congruents à une matrice diagonale dans cette base. Pour trouver cette base, on recourt à la *réduction simultanée* : on introduit la matrice  $N = M_1^{-1}M_2$ . Elle est diagonalisable et admet des valeurs propres réelles, avec  $N = R^{-1}\Lambda R$ . En fait la base cherchée s'exprime en fonction des vecteurs propres normalisés  $R$  de  $N$ . De plus  $R$  est inversible car ses composantes  $(\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2)$  forment une base de  $\mathbb{R}^3$ . En posant  $P = R^{-1}$ , les deux tenseurs s'expriment comme suit :

$$M_i = P^T \begin{pmatrix} \langle \mathbf{v}_1, M_i \mathbf{v}_1 \rangle & 0 \\ 0 & \langle \mathbf{v}_2, M_i \mathbf{v}_2 \rangle \end{pmatrix} P, \text{ avec } \begin{cases} N = M_1^{-1}M_2 = R^{-1}\Lambda R \\ R = (\mathbf{v}_1, \mathbf{v}_2) \\ P = R^{-1} \end{cases} \quad (A.16)$$

Le tenseur résultant de leur intersection est donné par :

$$\mathbf{M}_{1 \cap 2} = \mathbf{P}^T \begin{pmatrix} \|\langle \mathbf{v}_1, \mathbf{M}_i \mathbf{v}_1 \rangle\|_\infty & 0 \\ 0 & \|\langle \mathbf{v}_2, \mathbf{M}_i \mathbf{v}_2 \rangle\|_\infty \end{pmatrix} \mathbf{P}, \text{ avec } \begin{cases} \mathbf{N} = \mathbf{M}_1^{-1} \mathbf{M}_2 = \mathbf{R}^{-1} \Lambda \mathbf{R} \\ \mathbf{R} = (\mathbf{v}_1, \mathbf{v}_2) \\ \mathbf{P} = \mathbf{R}^{-1} \end{cases} \quad (\text{A.17})$$

Ainsi l'unique métrique riemannienne associée à l'approximation de la variété ainsi qu'à l'erreur de  $\mathbf{u}$  au point  $x$  est donnée par :

$$\forall \mathbf{u}, \mathbf{v} \in \mathbb{R}^3, g_x(\mathbf{u}, \mathbf{v}) = \langle \mathbf{u}, J_x \mathbf{P}^T \mathbf{D} \mathbf{P} J_x^T \mathbf{v} \rangle. \quad (\text{A.18})$$

avec :

- $J_x$  la matrice jacobienne permettant de passer du repère local au plan tangent de  $x$  dans la base canonique de  $\mathbb{R}^3$ .
- $\mathbf{P}$  la matrice des directions et qui correspond à la base commune des deux tenseurs pour laquelle ils sont congruents à une matrice diagonale, i.e  $\mathbf{M}_1 = \mathbf{P}^T \mathbf{D}_1 \mathbf{P}$  et  $\mathbf{M}_2 = \mathbf{P}^T \mathbf{D}_2 \mathbf{P}$ .
- $\mathbf{D}$  la matrice diagonale relative aux densités  $\rho_i$  associées aux directions  $\mathbf{v}_i$ . Elle encode la prescription de taille  $h_i = \rho_i^{-1/2}$  la plus restrictive en direction de  $\mathbf{v}_i$ .

REMARQUES. À noter que l'intersection n'est pas commutative comme illustré sur la figure A.3. En effet, le tenseur résultant dépend de  $\mathbf{N}$  défini par  $\mathbf{M}_1^{-1} \mathbf{M}_2$  ou  $\mathbf{M}_2^{-1} \mathbf{M}_1$  selon l'ordre dans lequel les champs de tenseurs sont traités et qui doit donc être fixé. Notons enfin que l'intersection est aussi utile dans le cas de plusieurs champs de solution  $\mathbf{u}_i$  (températures, pression, vitesse etc.) ainsi que pour la gradation.

#### A.2.4 Interpolation des tenseurs

Afin de définir une variété riemannienne continue  $(\Gamma, g)$ , nous devons désormais définir une manière d'interpoler la métrique  $g_p$  associée à tout point  $p$  de la variété idéale  $\Gamma$ , ce qui est utile pour les noyaux de raffinement et de lissage.

TRAVAUX CONNEXES. Pour toute arête de la triangulation, la métrique associée à un point d'une géodésique  $\gamma$  sous-tendue par cette arête est idéalement déterminée par combinaison convexe des valeurs de  $g$  sur les  $k+1$  points de contrôle de la B-spline associée à  $\gamma$ . Le problème c'est que cela nécessite  $k-1$  calculs intermédiaires pour obtenir  $g_p$  pour tout  $p \in \gamma$ , ce qui est très coûteux car en  $O(nk)$  pour une vague de raffinement par exemple.

En pratique, la métrique  $g_x$  est souvent interpolée à l'ordre un par le biais d'une *réduction simultanée* comme dans MMGS et MMG3D, dont le principe est expliqué à la section A.2.3. Dans ce cas, il y a plusieurs choix possibles selon la progression souhaitée de la prescriptions de taille  $h_{i,x}$  ou de la densité  $\rho_{i,x}$  associée à chaque direction  $\mathbf{v}_i$  de  $g_x$ , et qui peut être linéaire, géométrique, harmonique.

NOS CHOIX. Dans notre cas, nous décidons d'utiliser une interpolation directe des  $g_x$  par le biais du paradigme *log-euclidien* à l'instar de [78, 34]. Le réel avantage c'est qu'elle est commutative et respecte le principe du maximum [94]. Dans notre cas, le tenseur associé à tout point situé sur la courbe  $\gamma$  sous-tendue par une arête  $[pq]$  est donné par :

$$g_{\gamma[t]} = \exp [(1-t) \ln(\tilde{g}_{\gamma[0]}) + t \ln(\tilde{g}_{\gamma[1]})], \text{ où } \ln(g_x) = \mathbf{R}^T \begin{pmatrix} \ln |\lambda_1| & 0 \\ 0 & \ln |\lambda_2| \end{pmatrix} \mathbf{R} \quad (\text{A.19})$$

\*\*\*

## Étude d'algorithmes pour l'extraction de tâches

Rappelons nous que les noyaux de remaillage sont irréguliers et que le parallélisme inhérent est amorphe. Pour chaque noyau, nous formulons les tâches et leurs dépendances par un graphe, et les tâches compatibles sont ensuite extraites par le biais d'heuristiques adéquates. Afin d'extraire un parallélisme maximal tout en tenant compte des contraintes hardware, les heuristiques doivent être conçues avec soin. Dans ce chapitre, nous détaillons la démarche de recherche que nous avons initié pour le design de ces heuristiques.

---

B.1	Extraction de stable maximal . . . . .	141
B.1.1	En séquentiel . . . . .	141
B.1.2	Cas des méthodes gloutonnes . . . . .	142
B.1.2.1	Approches probabilistes . . . . .	142
B.1.2.2	Approches spéculatives . . . . .	143
B.1.3	Profiling et analyse expérimentale . . . . .	145
B.1.3.1	Paramètres de tests . . . . .	145
B.1.3.2	Sensibilité à l'ordonnancement . . . . .	147
B.1.3.3	Sensibilité au placement mémoire . . . . .	148
B.1.3.4	Ratio de conflits . . . . .	149
B.1.3.5	Qualité de la solution . . . . .	149
B.1.4	Synthèse . . . . .	151
B.2	Extraction d'un couplage maximal . . . . .	151
B.2.1	Cadre et motivations . . . . .	151
B.2.2	Cas d'un graphe non-biparti. . . . .	152
B.2.3	Synthèse . . . . .	154

---

PROBLÉMATIQUE. Le souci est que l'extraction d'un ensemble indépendant maximal (**maxindset**), le couplage de cardinalité maximale (**matching**) aussi bien que la coloration de graphes à nombre minimal de couleurs (**coloring**) sont des problèmes à la fois NP-complets et *irréguliers*. Bien qu'il existe des heuristiques gloutonnes triviales en séquentiel, en fournir une version parallèle efficiente reste ardue en raison :

- des dépendances intrinsèques et inévitables entre les itérés : la solution partielle  $\mathcal{U}^{[t]}$  obtenue à l'itéré  $t$  dépend de  $\mathcal{U}^{[t-1]}$ , puisqu'il s'agit soit d'une complétion de  $\mathcal{U}^{[t-1]}$  (calcul de la solution initiale) soit d'une amélioration de  $\mathcal{U}^{[t-1]}$  (recherche locale) (voir [178] chapitre 2, [200] chapitre 4, et [181] pour un aperçu et une analyse détaillée de ces algorithmes).
- de leur forte irrégularité<sup>1</sup>. Ici le temps de calcul est dominé par les accès-mémoire puisque les instructions ne consistent qu'à l'accès et mise à jour de données nodales [211, 212]. De plus, le traitement s'effectue de voisinage en voisinage par le biais d'un parcours en largeur (**maxindset**, **coloring**) ou en profondeur (**matching**) d'abord. Ainsi cela entraîne de multiples indirections, et donc une faible réutilisation des données en cache : les pénalités liées à la latence-mémoire se ressentent de manière significative sur le temps de restitution, notamment en contexte NUMA.
- du déséquilibre significatif de charges lié aux tailles inégales des voisinages d'une part, et à leur temps d'accès en mémoire inégal d'autre part.

<sup>1</sup>bien plus que les noyaux de remaillage. Ce fait est accentué par une intensité arithmétique quasi-nulle.

En raison de leur irrégularité, les complexités théoriques de ces heuristiques reflètent rarement le nombre d’itérés ainsi que les temps de restitution observés en pratique, même asymptotiquement quand  $|G|$  est très grand. En effet l’accélération obtenue varie sensiblement selon la manière dont l’heuristique est implémentée (notamment l’ordre de traitement, l’entrelacement des accès aux données, ou encore les primitives de synchronisation utilisées). Dans un contexte de pérennisation de code, une vraie problématique industrielle concerne la portabilité des performances indépendamment de l’environnement d’exécution, des paramètres architecturaux ou encore de la politique d’ordonnancement [219]. Dans ce cadre, notre challenge est de fournir des solutions scalables pour l’extraction des tâches, compte-tenu des trois contraintes précédentes. À ce titre, nous détaillons les idées et les choix de nos heuristiques pour `maxindset` et `coloring` dans la section B.1, puis pour `matching` dans la section B.2.

## B.1 EXTRACTION DE STABLE MAXIMAL

Rappelons qu’en théorie des graphes, un *stable* est un ensemble de sommets d’un graphe  $G = (V, E)$  qui soient deux à deux non adjacents. Ici le but est de trouver une partition  $\mathbb{P} = (\mathcal{U}_k)_{k \in \mathbb{N}}$  de stables à partir de  $G$  par le biais de `maxindset` ou de `coloring`. Notons que ces deux problèmes sont mutuellement réductibles<sup>2</sup>. En effet un stable  $\mathcal{U}$  peut être obtenu par coloration de  $G$  puis extraction de  $\mathcal{U}^* = \arg \max_k |\mathcal{U}_k|$ . Inversement la partition  $\mathbb{P}$  peut être obtenue par extraction répétée des  $\mathcal{U}^{[t]}$  sur le sous-graphe induit par  $V - \mathcal{U}^{[t-1]}$  à l’itéré  $t$ . Ainsi nous ne faisons pas la distinction entre les deux. Pour ce qui suit, on note  $n = |V|$ ,  $m = |E|$ ,  $n_c = |\mathbb{P}|$  le nombre de couleurs utilisées,  $\Delta$  et  $\bar{\Delta}$  les degré max et moyen de  $G$ , et enfin  $\chi$  le nombre chromatique de  $G$ .

### B.1.1 En séquentiel

TRAVAUX CONNEXES. Ces deux problèmes ont été largement étudiés dans la littérature, chaque approche étant optimisée pour une classe de graphes ou d’applications données (voir les thèses décrites dans [200, 178] pour un aperçu de ces méthodes). À part les méthodes exactes, on peut distinguer trois familles d’heuristiques :

- *gloutonnes* : elles visent à construire ou améliorer qualitativement la solution  $\mathcal{U}^{[t]}$  à chaque itéré  $t$  par le biais d’une opération élémentaire.
- *avancées* : elles visent à chercher la solution optimale  $\mathcal{U}^*$  par parcours de l’espace de solutions. À l’itéré  $t$ , le choix de la solution candidate  $\mathcal{U}^{[t]}$  est déterminé de manière probabiliste tout en minimisant une fonctionnelle de coût.
- par *réduction* : l’instance est simplifiée, ou la recherche de stables est reformulée en un cas particulier d’un autre problème combinatoire (satisfiabilité de prédictats, recherche de cliques par exemple) que l’on sait résoudre efficacement (en temps polynomial).

Une synthèse de ces approches est donnée à la table B.1. Rappelons que l’extraction de tâches n’est qu’une étape parmi d’autres au sein d’un noyau de remaillage : son coût doit donc rester négligeable pour être applicable dans notre contexte. Ainsi l’heuristique retenue doit fournir un bon compromis entre qualité de la solution  $\mathcal{U}$ , complexité et nombre d’itérés avant convergence. Dans ce cadre, notre attention s’est portée sur les heuristiques gloutonnes en tirant profit du fait que  $G$  est planaire et que  $\Delta$  est borné.

APPROCHES GLOUTONNES. Partant du principe que les heuristiques simples sont souvent les plus efficaces<sup>3</sup> dans la plupart des cas (voir par exemple le cas du minimum-degree décrit dans [184] pour s’en convaincre), nous nous sommes intéressés aux approches gloutonnes. En effet, elles offrent la flexibilité nécessaire en vue d’une parallélisation efficiente<sup>4</sup>. Pour `maxindset`, l’approche gloutonne la plus simple consiste à rajouter successivement chaque sommet  $v \in V^{[t]}$  dans  $\mathcal{U}^{[t]}$  tout en supprimant ses voisins  $\mathcal{N}[v]$  des sommets restant à traiter  $V^{[t]}$ . Il convient pour un graphe  $G$  avec un faible degré puisque la taille du sous-graphe  $G' \subset G$  induit par  $V^{[t]}$  privé de  $\mathcal{N}[v] \cup \{v\}$  décroît avec moins de  $\Delta + 1$

<sup>2</sup>Il existe une réduction polynomiale entre les deux problèmes, ainsi résoudre l’un permet de résoudre l’autre.

<sup>3</sup>En terme de ratio entre qualité de la solution  $\mathcal{U}$  et complexité ou temps de calcul effectif.

<sup>4</sup>En terme de ratio entre accélération obtenue et surcoût induit.

Table B.1: Synthèse des approches séquentielles pour le calcul de stables.

approche	principe
gloutonnes	- ajout d'un sommet $u$ dans $\mathcal{U}^{[t]}$ ou suppression de $u$ de $G$ [184]
	- amélioration de $\mathcal{U}^*$ par parcours de l'espace de solutions. [178]
	- recherche locale avec sélection probabiliste des candidats. [185]
avancées	- recherche locale par recuit-simulé [179]
	- recherche locale par une machine de Boltzmann. [180]
	- recherche locale par des noyaux évolutionnaires. [186]
	- recherche locale et évitement de cycle par prohibition. [187]
par réduction	- suppression de sommets singuliers, isolés ou jumeaux. [188]
	- réduction vers la satisfiabilité de prédicts. [189]
	- réduction vers une recherche de clique. [190]
	- réduction vers un couplage de graphes. [191]

sommets à chaque étape. Ainsi la taille du stable  $\mathcal{U}$  vaut au moins  $\lceil \frac{n}{\Delta+1} \rceil$ . Une preuve est d'ailleurs donnée dans [201].

Pour coloring, l'approche la plus simple et rapide est **First-fit**. À chaque itéré, elle consiste à choisir un sommet  $v \in V$ , et à affecter la plus petite couleur admissible à  $v$ , c'est-à-dire celle qui n'a pas encore été assignée à un de ses voisins. Le ratio d'approximation  $\rho = \frac{n_c}{\Delta}$  ainsi que le nombre d'opérations sont proportionnels au degré de chaque sommet. Ainsi le coût de l'heuristique est  $\sum_i \deg[v_i] = O(m)$ . Une extension de First-fit concerne les approches **Degree-based ordering** : ici le choix du prochain voisin  $u \in N[v]$  est basé sur son degré. Si  $\Delta$  est suffisamment petit alors  $|P| \leq \Delta + 1$ . Selon le critère choisi on distingue trois variantes dans la littérature :

- le *largest-degree*: on choisit le voisin de degré maximal à chaque itéré [192].
- le *saturation-degree* : le critère de sélection correspond à l'indice de saturation qui est défini par le nombre de voisins d'un sommet  $v$  ayant une couleur différente de  $v$  [193].
- l'*incidence-degree* : on considère juste le nombre de voisins déjà colorés [194].

### B.1.2 Cas des méthodes gloutonnes.

Rappelons que le calcul d'un stable  $\mathcal{U}$  ou d'une partition de stables  $P = (\mathcal{U}_k)_{k \in \mathbb{N}}$  n'est qu'une étape préliminaire de chaque noyau, et qu'il est crucial que son coût reste négligeable, surtout pour le noyau de contraction. D'un autre côté, si on veut extraire un degré de parallélisme élevé, il faudrait que la qualité de la solution  $\mathcal{U}^*$  reste acceptable. En vue de construire un algorithme d'extraction de stable qui concilie ces contraintes, nous nous sommes intéressés de près aux approches parallèles gloutonnes en vue d'identifier leurs points forts ainsi que leurs points faibles d'un point de vue théorique et pratique.

#### B.1.2.1 Approches probabilistes

**PRINCIPE.** Remarquons que les approches précédentes impliquent des dépendances d'itérations inévitables, puisque le stable  $\mathcal{U}^{[t]}$  est construit par enrichissement de  $\mathcal{U}^{[t-1]}$  à partir de  $G^{[t]}$  à l'itéré  $t$ . Ainsi une manière efficace de "casser" ces dépendances réside dans une sélection *aléatoire* ou *probabiliste* du prochain voisin à rajouter dans  $\mathcal{U}$ . Parmi les approches parallèles existantes, la plus populaire et la plus utilisée est celle de Luby [182] ainsi que ses extensions [réf]. Il s'agit d'un algorithme probabiliste structuré de manière synchrone en rounds. Ainsi chaque sommet actif  $v$  est ajouté selon une probabilité  $P[v]$  relative au nombre de voisins actifs, et donc de son degré, avec  $P[v] = 1/2\deg[v]$ . Si deux voisins  $u$  et  $w$  sont sélectionnés dans le même round  $t$ , alors le sommet de degré minimal est retiré de  $\mathcal{U}^{[t]}$ . Une fois le stable  $\mathcal{U}^{[t]}$  extrait à l'itéré  $t$ , on construit le sous-graphe  $G^{[t]} \subset G^{[t-1]}$  induit par  $V^{[t]}$  privé de  $\mathcal{U}^{[t]} \cup N[\mathcal{U}^{[t]}]$ , et on réitère la procédure jusqu'à ce qu'il n'y ait plus de sommets à traiter, c'est à dire lorsque  $|V^{[t]}| = 0$ , ce qui se fait en  $O(\log n)$  rounds avec  $O(m)$  cores. Notons juste qu'il s'agit d'une approche Monte-Carlo dans la mesure où la probabilité que la solution calculée  $\mathcal{U}^*$  soit

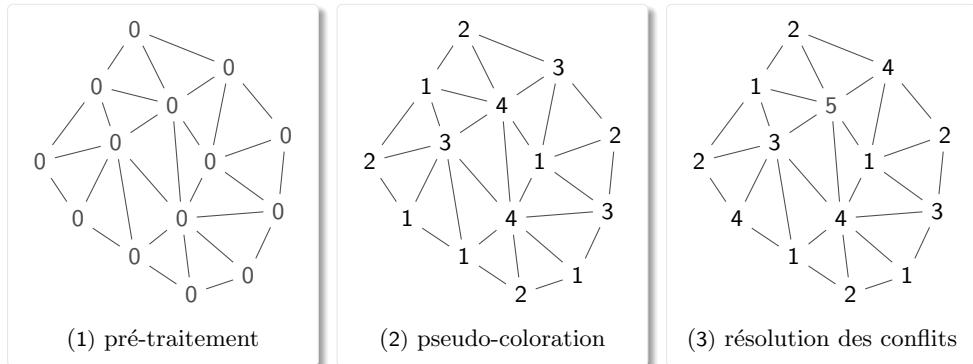


Figure B.1: Principe de l'extraction de stables d'un graphe par une approche spéculative. Une partition initiale de stables  $\mathbb{P} = (\mathcal{U}_k)_{k \in \mathbb{N}}$  est d'abord calculée dans une phase de pseudo-coloration sans se soucier des data-races. Les conflits de couleur sont ensuite résolus dans une vague à part. Le stable de cardinalité maximale peut être extrait avec  $\mathcal{U}^* = \arg \max |U_k|_{k \in \mathbb{N}}$ . Les heuristiques diffèrent ensuite sur la manière de traiter chaque phase.

incorrecte est non nulle [182]. L'approche de Luby est intéressante en raison de ses bonnes propriétés de convergence. Néanmoins nous ne l'avons pas retenue car à chaque itéré  $t$  :

- elle nécessite la régénération des graines aléatoires pour converger d'une part, ce qui peut être très coûteux en cycles quand  $|V|$  et  $p$  sont très grands;
- le calcul du sous-graphe  $G^{[t]} = (V^{[t]}, E^{[t]})$  induit par  $V^{[t-1]}$  privé de  $\mathcal{U}^{[t]} \cup \mathcal{N}[\mathcal{U}^{[t]}]$  peut être couteuse selon la manière dont on l'implémente d'autre part car l'extraction des arêtes  $E^{[t]} = \{(u, v) \in E^{[t-1]}, u, v \in \mathcal{U}^{[t]}\}$  implique une intersection ensembliste et donc de multiples comparaisons et recopies de données.

In fine, l'efficacité de l'algorithme dépend concrètement de la manière dont on l'implémente ce qui réduit sa portabilité. Une version déterministe est également proposée dans [182], mais elle est encore une fois trop coûteuse pour être applicable dans notre contexte, car elle implique notamment le calcul d'un grand nombre premier  $q$  tel que  $n^{[t]} \leq q \leq 2n^{[t]}$  à chaque round  $t$ .

### B.1.2.2 Approches spéculatives

**MOTIVATIONS.** Ainsi malgré ses bonnes propriétés de convergence, une approche Monte-Carlo reste encore inadaptée à notre contexte. En recherche d'une autre alternative, nous nous sommes tournés vers les approches **spéculatives** pour le calcul d'une partition de stables par le biais de **coloring**. Il s'agit d'algorithmes parallèles structurés en rounds (synchrone) et basés sur un schéma d'exécution optimiste : une pseudo-solution  $\tilde{\mathcal{U}}$  est d'abord calculée, et les erreurs sont ensuite résolues dans une étape à part comme illustré sur la figure B.1. Ici, le caractère synchrone permet de mieux structurer la gestion des contentions d'accès en mémoire partagée d'une part, et d'analyser la convergence de ces algorithmes d'autre part.

À l'instar des approches probabilistes, les approches spéculatives sont également basées sur des heuristiques gloutonnes. Parmi les travaux plus récents [204–206, 211], un intérêt particulier est donné à **First-fit** en raison de son efficacité en terme de ratio entre qualité de la solution  $\mathcal{U}$  et complexité ou nombre de rounds. En vue d'inférer notre propre heuristique, nous nous sommes intéressés en particulier à trois d'entre elles.

**GEBREMEDHIN ET AL.** La première version spéculative de **First-fit** a été initiée par Gebremedhin dans [205, 200], et est basée sur une partition de  $G$  par blocs. L'idée est d'assigner une couleur en parallèle sans se soucier des data races dans une phase dite de **pseudo-coloration**, puis de résoudre les conflits – c'est à dire deux voisins ayant la même couleur – dans une phase de **correction**. Pour cela, le graphe est partitionné en  $k$  blocs  $V_i$  de tailles égales  $|V_i| \approx \lceil \frac{n}{p} \rceil$  qui seront ensuite assignés aux  $k$

threads punaisés sur  $k$  cores. Pour chaque sommet, on choisit systématiquement la plus petite couleur admissible par le biais de First-fit. La phase de correction est scindée en trois étapes synchrones<sup>5</sup> :

- chaque stable  $\mathcal{U}_j \in \mathbb{P}$  est re-partitionné en  $k$  blocs, puis on procède à leur pseudo-coloration.
- les sommets conflictuels restants sont ensuite identifiés et indexés dans une table  $R$ ;
- les sommets  $v_i \in R$  sont ensuite colorés en séquentiel.

Ici la majorité des sommets de  $G$  est correctement coloré à l'issue de la pseudo-coloration qui s'effectue de manière complètement asynchrone<sup>6</sup>. À l'issue de cette phase, le nombre de sommets conflictuels est borné par  $\frac{\bar{\Delta}}{2}(p-1)$  et est donc indépendante de  $n = |G|$ . Durant la correction, le nombre de sommets restants à traiter en séquentiel est borné par  $\frac{2}{n}p^2\Delta^2$  et décroît significativement quand le ratio  $\frac{n}{p}$  est grand.

ÇATALYUREK ET AL. En dépit du faible ratio de conflits, l'approche précédente comporte une étape séquentielle relative à la phase de correction. En notant  $r$  le ratio du temps d'exécution relative à cette étape sur le temps de restitution, son accélération est bornée par  $\frac{1}{r}$  quelque soit le nombre  $p$  de cores, d'après la loi d'Amdahl. Dans ce cadre, une extension de l'algorithme a été développée par Çatalyurek *et al.* dans [204], et dédiée aux machines manycore et aux architectures massivement parallèles. Ici l'idée est d'éliminer l'étape séquentielle relative à la correction. Ainsi l'algorithme ne comporte plus que deux phases synchrones relatives à :

- la pseudo-coloration : chaque thread maintient un tableau local `forbidden` associant à chaque sommet  $v$  les couleurs qui lui sont interdites car déjà attribuées à ses voisins.
- la détection de conflits : les threads réexaminent les couleurs d'un sous-ensemble de sommets marqués, et qui vont ensuite être recolorés à l'itération d'après. En cas de conflit, le sommet voisin de plus petit ID est marqué comme étant à recolorer et placé dans une liste de tâches  $R$ .

Dans sa version générique, il s'agit d'une approche itérative contrairement à la précédente. Elle ne nécessite aucun partitionnement de  $G$  mais implique deux barrières de synchronisation par itéré. Notons qu'une version dataflow a été conçue spécialement pour le portage de l'algorithme sur une machine Cray XMT nécessitant une synchronisation à granularité plus fine (pas de barrière mais des instructions dédiées `purge`, `readff`, `readfe`, `writelf`, voir [220, 221]), dans le but d'assigner un maximum de couleurs correctes et définitives aux sommets dès le premier itéré.

Aucune étude théorique n'est fournie quant à la convergence et l'efficacité. Néanmoins le nombre constaté de conflits reste négligeable comparé au ratio  $\frac{n}{p}$ . De plus, le nombre d'itérés ne croît que de manière logarithmique en  $p$  tandis que le nombre  $|R|$  de sommets à traiter décroît de manière exponentielle à chaque itéré.

ROKOS ET AL. Malgré une bonne vitesse de convergence et une bonne scalabilité sur des machines NUMA classiques (dual-socket Intel Nehalem et Sun Niagara avec 4 et 8 cores par socket) et massivement manycore (Cray XMT avec 128 noeuds reliés par un DSM et 128 cores par noeud), l'approche précédente implique une contrainte non négligeable : elle nécessite deux barrières par itéré. Ainsi les pénalités liées au déséquilibre de charges entre les  $p$  cores croît proportionnellement à  $p$  et au nombre d'itérés. Dans l'optique d'en fournir une version plus asynchrone, une extension a été développée par Rokos *et al.* dans [206]. Dans ce cadre, quand un sommet s'avère défectueux, il est immédiatement recoloré au lieu de reporter le traitement à l'itéré suivant, ce qui permet de lever la seconde barrière de synchronisation.

Ici encore aucune garantie théorique n'est donnée quant à la convergence contrairement à [205]. Le gain effectif par rapport à l'approche précédente est nuancé avec une accélération relative de 1.35 et un nombre d'itérés sensiblement identique sauf pour les instances de graphes très irrégulières telles que RMAT-B [207]. En effet, on constate un ratio important de sommets correctement colorés dès la première phase (près de 70%) : ainsi le surcoût induit par le déséquilibre est due en grande partie à la première barrière qui elle est inévitable. De plus, la suppression de la seconde barrière induit un comportement

<sup>5</sup>Les étapes sont séparés par des barrières de synchronisation.

<sup>6</sup>Elle ne requiert ni verrous, ni points de synchronisation.

inattendu sur les architectures SIMD : l'algorithme peut tomber dans une boucle infinie et ne jamais se terminer. C'est typiquement ce qui se passe lorsque deux threads ont systématiquement la même vision de la plus petite couleur admissible car ils prennent les décisions au sein d'un même cycle CPU.

Table B.2: Synthèse d'approches gloutonnes parallèles pour le calcul de stables.

approche	avantages	inconvénients
Luby	✓ bonne convergence en $O(\log n)$ ✓ faible probabilité que $\mathcal{U}^*$ incorrecte	✗ régénération coûteuse de graines. ✗ calcul coûteux du sous-graphe induit.
Gebremedhin	✓ pseudo-coloration asynchrone ✓ nombre de conflits indépendant de $n$ ✓ routine non itérative.	✗ résolution séquentielle de conflit. ✗ ordonnancement figé. ✗ phase de correction multi-étapes.
Catalyurek	✓ résolution parallèle de conflits. ✓ nombre de conflits $n_c$ négligeable en $\frac{n}{p}$ . ✓ granularité fine et convergence rapide ✓ version dataflow massivement parallèle	✗ routine itérative. ✗ pas de borne théorique sur $n_c$ ✗ deux barrières par itéré : déséquilibre. ✗ version dataflow non portable
Rokos	✓ une seule barrière par itéré ✓ faible nombre d'itérés $t_{\max}$ en pratique. ✓ efficace sur les graphes irréguliers	✗ pas de borne théorique sur $n_c$ et $t_{\max}$ . ✗ terminaison non garantie si SIMD. ✗ asynchrone : irrégularité accrue

### B.1.3 Profiling et analyse expérimentale

MOTIVATIONS. Bien que respectivement construites de manière extensive, chacune des approches décrites dans la section B.1.2 présentent des avantages et inconvénients particuliers récapitulés à la table B.2. Ainsi il est impossible d'affirmer de manière absolue si une heuristique est une meilleure qu'une autre car cela dépend à la fois du contexte d'exécution et des contraintes que l'on priviliege. En effet, ils peuvent avoir un comportement très différent pour une instance de graphe, un environnement d'exécution, une politique d'ordonnancement ou une architecture donnés, compte-tenu de leur irrégularité. Ainsi nous avons décidé de mener une étude expérimentale et comparative en vue d'inférer :

1. leur sensibilité vis-à-vis de la topologie et de l'irrégularité du graphe en entrée.
2. leur scaling et leur sensibilité à la politique d'ordonnancement appliquée.
3. leur sensibilité à la localité mémoire et à l'affinité thread-core, notamment en contexte NUMA.
4. le ratio de conflits  $\frac{n_c}{n}$  et leur évolution en fonction du nombre de cores  $p$ .
5. la qualité de la solution retournée<sup>7</sup>, et sa dégradation par rapport à la version séquentielle.

Pour cela, nous avons implémenté chacune de ces approches en C++11 et OpenMP4 et procédé au profiling des codes correspondants. Nous les avons ensuite testés sur trois instances de graphes irréguliers RMAT et exécutés sur une machine dual-socket Intel Skylake dont la topologie est décrite à la figure B.2.

#### B.1.3.1 Paramètres de tests

GRAPHES DE TESTS. Rappelons nous que le graphe de tâches associé à un noyau est un sous-graphe induit du graphe primal ou dual de la triangulation. À ce titre, il peut avoir une topologie quelconque et un degré maximal élevé. Remarquons que dans notre contexte d'un maillage anisotrope, le degré d'un point peut être relativement élevé, à cause des étirements des mailles selon une direction privilégiée. Pour des tests consistants, nous avons ainsi opté pour un modèle de graphe plus général RMAT [207], dont la construction et les paramètres sont décrits au paragraphe suivant. Partant du principe que si une heuristique est efficace pour une instance irrégulière alors il sera au moins aussi efficace pour une instance plus régulière (l'inverse n'est toutefois pas vraie).

<sup>7</sup>  $|\mathcal{U}^*|$  pour maxindset et  $|\mathbb{P}|$  pour coloring

Table B.3: Caractéristiques de graphes de tests pour le calcul de stables.

graphe	$ V $	$ E $	deg max	deg moy	variance	$\omega_{(1,1)}$	$\omega_{(1,2)}$	$\omega_{(2,1)}$	$\omega_{(2,2)}$
RMAT-ER	$16 \cdot 10^6$	$128 \cdot 10^6$	51	9.16	21.23	0.25	0.25	0.25	0.25
RMAT-G	$16 \cdot 10^6$	$128 \cdot 10^6$	606	3.13	22.17	0.45	0.15	0.15	0.25
RMAT-B	$16 \cdot 10^6$	$128 \cdot 10^6$	5398	1.72	123.17	0.55	0.15	0.15	0.15

Ici chaque graphe de test est généré par subdivision récursive de la matrice d'adjacence associée en 4 quadrants (1,1), (1,2), (2,1) et (2,2). Les arêtes sont réparties dans les quadrants selon une certaine probabilité. Cette distribution est déterminée par quatre coefficients  $\omega_{(1,1)}$ ,  $\omega_{(1,2)}$ ,  $\omega_{(2,1)}$  et  $\omega_{(2,2)}$  correspondant à leur densité de probabilité. Après initialisation des coefficients  $\omega_{(i,j)}$ , l'algorithme choisit de placer une arête dans l'un des quadrants suivant selon leurs valeurs. Le quadrant choisi est ensuite subdivisé en quatre sous-quadrants et la procédure est itérée jusqu'à obtenir un quadrant unité ( $1 \times 1$ ). La génération des arêtes est itérée  $m = |E|$  fois pour obtenir G. En choisissant les 4 paramètres judicieusement, on peut générer des graphes avec des caractéristiques particulières. Pour nos tests, on choisit de reprendre les mêmes que dans [204] dont les détails sont donnés à la table B.3. De manière concrète, les instances ont été générées par le biais de l'outil open-source PaRMAT décrit dans [222] et disponible sur <https://github.com/farkhor/PaRMAT>.

**PROFILING.** Le profiling a été effectué sur une machine 48-core dual-socket Intel Skylake (deux processeurs Intel Xeon Platinum 8168 avec 24 cores par socket cadencés à 2.7 Ghz). Les threads sont placés explicitement sur les cores en round-robin, tandis que l'hyperthreading est désactivé afin de réduire les contentions d'accès au bus et pour disposer de plus de mémoire par thread. La hiérarchie mémoire comporte trois niveaux de cache et une RAM de 383 Go partagée par les deux processeurs comme illustré sur la figure B.2. Elle est structurée en 2 noeuds NUMA avec un cache L3 de 33 Mo par noeud. Chaque core dispose d'un cache L2 à 1 Mo et de deux caches L1 (données et instructions) à 32

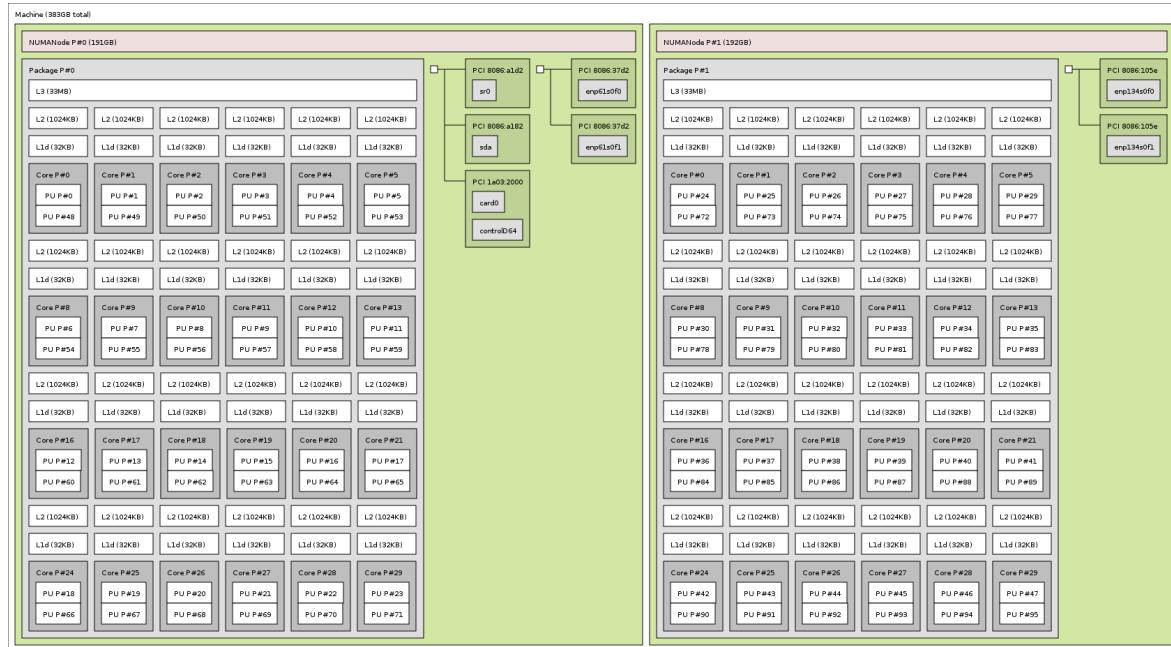


Figure B.2: Topologie de la machine 48-core dual-socket utilisé pour nos tests. La mémoire est logiquement décomposée en 2 noeuds NUMA avec 24 cores Intel Skylake partageant un cache L3 de 33 Mo chacun. Ici chaque core dispose d'un cache L2 de 1 Mo, outre deux caches L1 (données et instructions) de 32 Ko chacun, et peut être hyperthreadé avec 2 PU par core. La topologie est générée par l'outil open-source hwloc de l'INRIA [177].

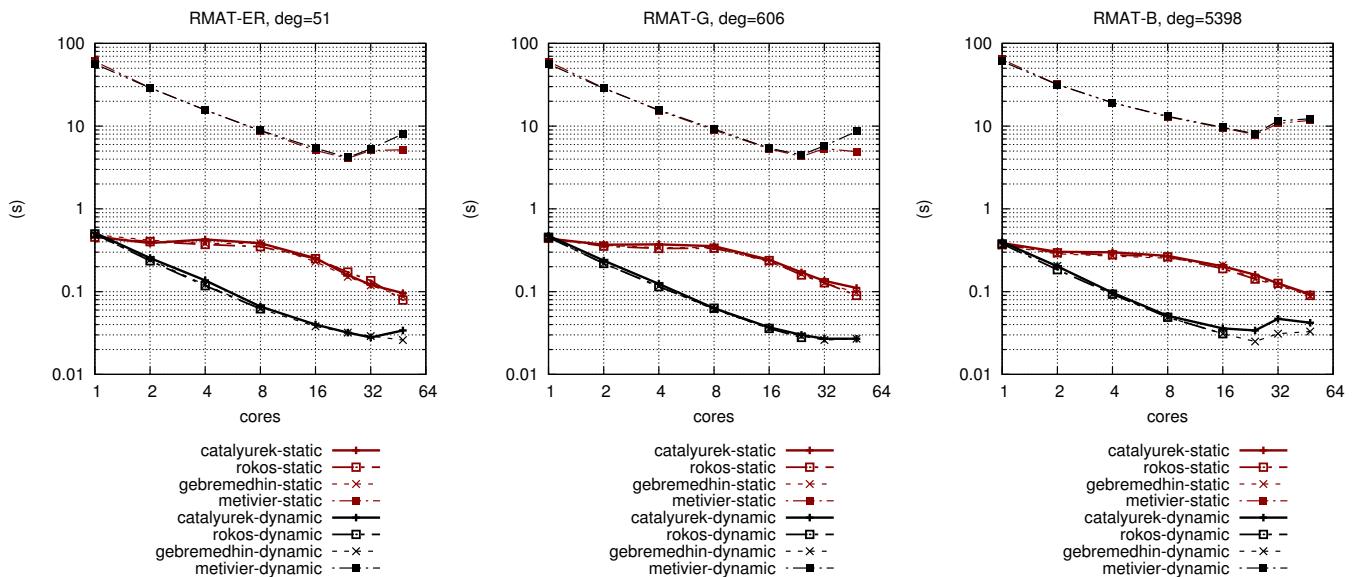


Figure B.3: Temps de restitution selon les politiques d'ordonnancement appliquées.

Ko chacun. Le code a été compilé avec le compilateur C++ d'Intel icpc version 17 avec les optimisations activés `-O3 -march=native` incluant la vectorisation automatique par le biais des instructions SIMD (extensions SSE4). Rappelons juste qu'il est essentiel de connaître la topologie de la machine car on conçoit un algorithme parallèle pour une topologie dédiée (anneau, grappe etc.).

Nous utilisons les primitives Posix-conforme `std::chrono::high_resolution_clock` de la bibliothèque standard C++11, afin d'obtenir des mesures précises. Enfin pour limiter les biais de mesure, on effectue 5 runs par heuristique (voir table B.2) et par graphe (voir table B.3), et on considère la moyenne sur chacune des métriques considérées (voir page 145). Dans le cas d'un ordonnancement dynamique, la taille minimale des blocs est fixée à  $k = |\mathcal{V}_i| = 32\,000$  soit 0.002% de  $|\mathcal{V}|$  afin d'avoir une granularité suffisamment fine.

### B.1.3.2 Sensibilité à l'ordonnancement

GRANULARITÉ. Le scaling du temps de restitution pour chaque heuristique de la table B.2 et pour chaque politique d'ordonnancement est illustrée à la figure B.3. Les résultats obtenus confirment notre intuition stipulant qu'une approche Monte-Carlo serait inadaptée en raison du surcoût lié à la régénération des poids aléatoires : ici elle est presque 100 fois plus couteuse comparé à une approche spéculative. Un second constat concerne la sensibilité des performances des heuristiques spéculatives à l'ordonnancement appliqué, ce qui est une caractéristique des problèmes irréguliers. Ici le choix de la granularité, c'est-à-dire la taille des blocs assignés à chaque thread, est crucial : s'il est trop grossier alors on paie les pénalités liées au déséquilibre des charges (c'est le cas d'un ordonnancement statique avec des blocs de taille  $k = \lceil \frac{n}{p} \rceil$ ), tandis que s'il est trop fin alors on paie le surcoût lié à la synchronisation des threads pour l'acquisition de tâches.

Au vu de l'écart d'accélération constaté entre l'application d'un ordonnancement statique et dynamique avec une granularité de 0.002% de  $n$ , on conclut que le déséquilibre de charges est le facteur limitatif le plus significatif à la scalabilité des heuristiques. De plus, l'accélération obtenue n'est significative qu'à partir de 16 threads lors d'un ordonnancement statique, ce qui est normal puisque le déséquilibre tend à diminuer quand  $p$  croît. Notons toutefois que ce constat n'est pas vérifié pour l'heuristique probabiliste, qui elle est insensible à l'ordonnancement appliqué. En fait la différence s'explique par le fait qu'elle est plus compute-intensive que ses variantes spéculatives : dans le premier cas le coût en cycles CPU de chaque tâche est certes plus important mais aussi plus uniforme, tandis que dans le dernier cas il est dominé par le cout des accès-mémoire qui lui varie sensiblement selon le degré des sommets. Notons enfin que le temps de restitution est sensiblement égal pour les trois

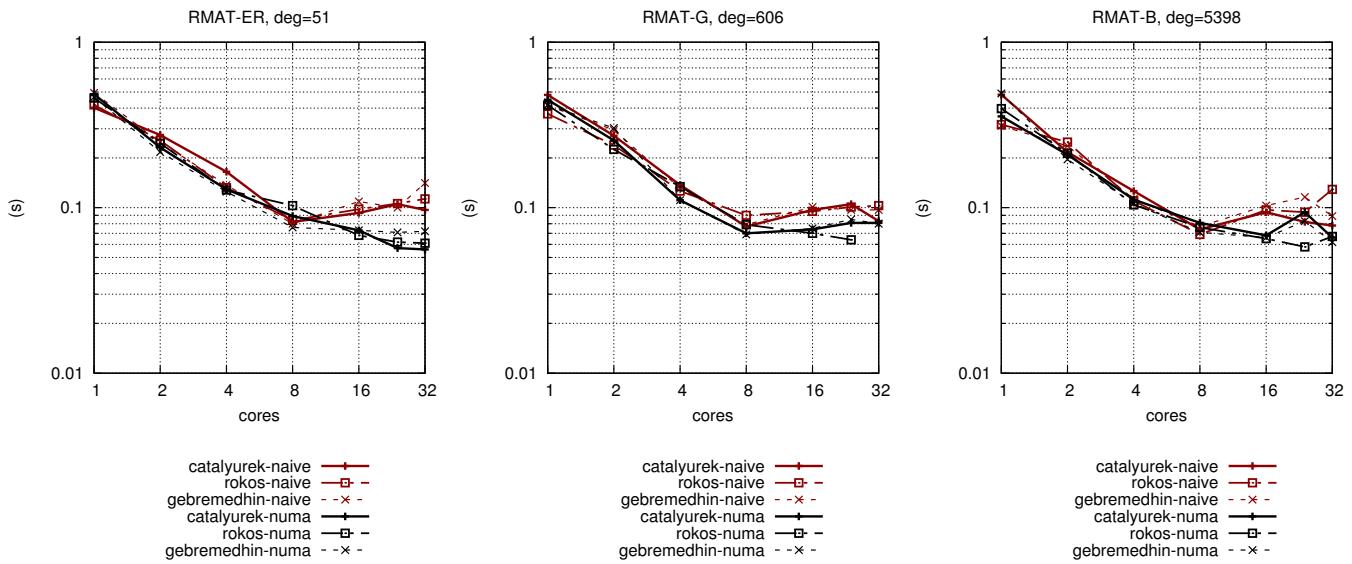


Figure B.4: Temps de restitution selon le placement mémoire appliquée.

instances de RMAT malgré un ratio de conflits complètement différent (voir figure B.5). Cela montre l’efficacité de ces approches spéculatives.

### B.1.3.3 Sensibilité au placement mémoire

EFFETS NUMA. Afin d’évaluer l’impact de la localité des accès-mémoire sur la scalabilité, nous avons profilé le temps de restitution des heuristiques selon deux politiques de placement à la figure B.4. Dans le premier cas dit naïf, la mémoire partagée est exclusivement allouée sur le noeud #1 par le biais de l’outil Linux `numactl`, tandis que dans le second cas dit NUMA-aware, l’allocation est répartie sur les deux noeuds par le principe du first-touch : lorsqu’un thread effectue un `malloc`, la page mémoire associée à l’adresse demandée ne lui est réellement allouée qu’au moment où elle est écrite pour la première fois. Dans ce cas, les threads procèdent à une vague d’initialisation de sorte que chaque blocs de données soit alloué sur la mémoire physique la plus proche des cores sur lesquels ils sont punaisés, pour chaque structure de données en mémoire partagée (notamment le graphe RMAT, ainsi que les tableaux `color` et `tasklist`). Ici une politique d’ordonnancement dynamique est appliquée avec une granularité de 0.002%. Pour des résultats plus significatifs, nous avons exceptionnellement effectué le profiling sur une machine 32-core dual-socket Intel Haswell structuré en 4 noeuds NUMA cette fois, avec 8 cores cadencés à 2.5 Ghz par noeud et 16 cores par socket (voir figure ??).

Nous constatons une variance significative du temps de restitution dans le cas d’un placement naïf comparé à la version NUMA-aware, au fur et à mesure qu’on augmente le nombre de cores  $p$ . En effet en raison de l’ordonnancement dynamique appliqué, les threads peuvent se voir attribuer des tâches allouées sur des pages mémoire physiquement éloignées du socket auquel ils sont punaisés, et cela de manière complètement imprédictible. Ainsi malgré la grande capacité du cache L3, ils souffrent quand même de la latence liées à ces accès distants. Remarquons que ce constat est beaucoup plus marqué au delà de 8 cores et surtout à partir de 16 cores : nous assistons à un effet NUMA. Dans ce cadre, l’accélération par core est résorbée par les pénalités liées aux accès-mémoire distants, outre le cout de synchronisation des threads relatif à l’ordonnancement dynamique appliqué. À noter que cet effet est beaucoup plus prononcé dans le cas d’un placement naïf.

Enfin notons que les comportements des trois heuristiques sont sensiblement identiques puisqu’elles disposent de patterns d’accès-mémoire similaires. Plus que de les comparer entre elles, ici le but réel est d’identifier les facteurs limitatifs à leur scalabilité en raison de leur irrégularité. Dans notre contexte, cela nous donnera les pistes pour la structuration des requêtes topologiques par les noyaux de remaillage.

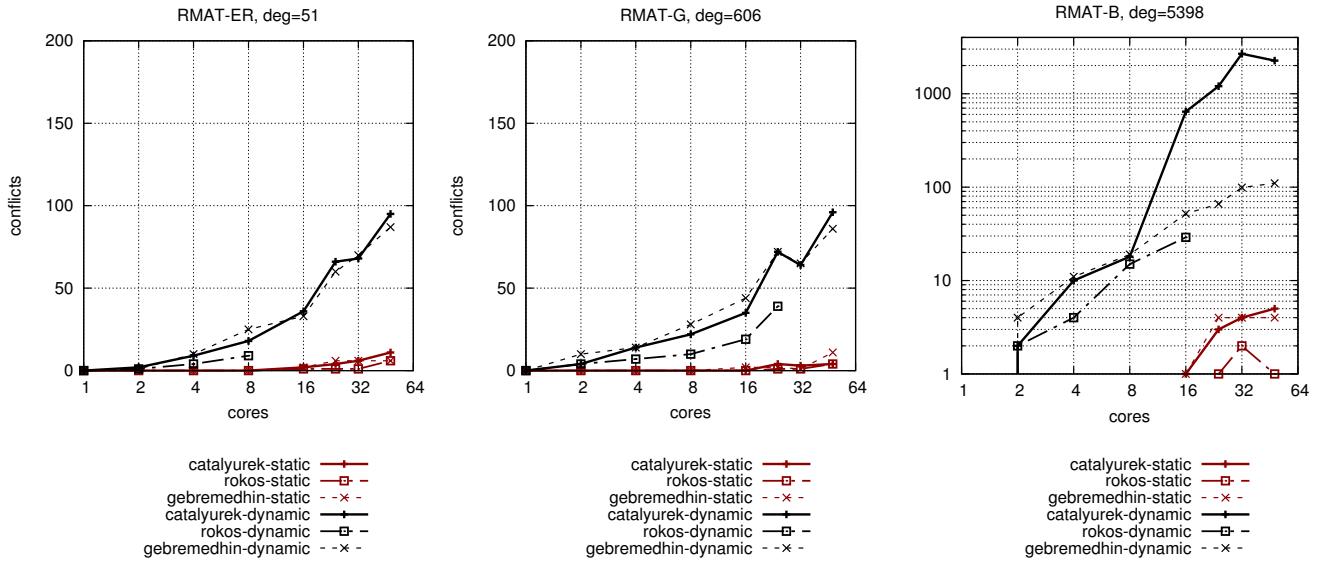


Figure B.5: Nombre de conflits détectés et résolus par graphe et par heuristique.

#### B.1.3.4 Ratio de conflits

RATIO DE CONFLITS. Le scaling du nombre de conflits  $n_c$  résolus pour chaque heuristique spéculative sur chaque graphe et pour chaque politique d'ordonnancement est donné à la figure B.5. Une première constatation est que  $n_c$  est négligeable devant la taille  $n = |V|$  du graphe : en effet le ratio  $r_c = \frac{n_c}{n}$  n'excède même pas les 1% quelque soit le graphe ou l'ordonnancement appliqué. Notons que les conflits ne se produisent qu'aux interfaces de blocs de sommets attribués aux cores. Ici on constate qu'ils évoluent proportionnellement au nombre de cores  $p$ , et varient sensiblement selon l'ordonnancement :

1. **statique** : dans ce cas les sommets sont répartis en  $p$  blocs de tailles  $\lceil \frac{n}{p} \rceil$  attribués aux  $p$  cores. En fait  $r_c$  évolue linéairement au ratio de sommets aux interfaces sur  $n$  qui tend asymptotiquement vers zéro : ainsi  $r_c$  est quasiment nul.
2. **dynamique** : cette fois le graphe est scindé en  $k \cdot n$  blocs de taille  $k$ , et chaque core se voit attribuer  $\lceil \frac{n}{k \cdot p} \rceil$  blocs. Comme le nombre de blocs croît, le nombre de sommets aux interfaces croît proportionnellement, et il en va de même pour  $r_c$ .

En outre, on constate que le scaling de  $r_c$  varie significativement selon l'irrégularité du graphe. Bien qu'il soit sensiblement égal pour RMAT-ER et RMAT-G, il croît de manière exponentielle pour RMAT-B – l'instance la plus irrégulière avec  $\Delta = 5\,398$  et une variance de 124 – notamment dans le cas d'un ordonnancement dynamique. Ici le ratio de conflits est légèrement meilleur pour l'approche de Gebremedhin puisque la phase de correction n'est pas itérée contrairement aux deux autres. Notons toutefois que cet écart n'est pas si significatif que cela car la plupart des conflits ont lieu lors du premier itéré.

#### B.1.3.5 Qualité de la solution

QUALITÉ DE LA SOLUTION. Afin de comparer la qualité des solutions fournies par les heuristiques ainsi que leur scaling, nous avons profilé le ratio de taille des stables  $\mathcal{U}^*$  sur  $n$ , ainsi que le nombre effectif de couleurs  $|\mathbb{P}|$  pour chaque instance de RMAT à la figure B.6. Ici on constate d'emblée que  $|\mathcal{U}^*|$  et  $|\mathbb{P}|$  restent invariantes pour chaque heuristique, qu'importe le nombre  $p$  de threads ou l'ordonnancement appliqué. Notons qu'obtenir des solutions de qualité identique au cas séquentiel, pour toute instance fournie en entrée, reste un challenge dans le design d'une heuristique parallèle. Ici ces résultats montrent l'efficacité de ces trois approches. Un second constat concerne la sensibilité des résultats par rapport à l'irrégularité du graphe et donc à la répartition des degrés des sommets.

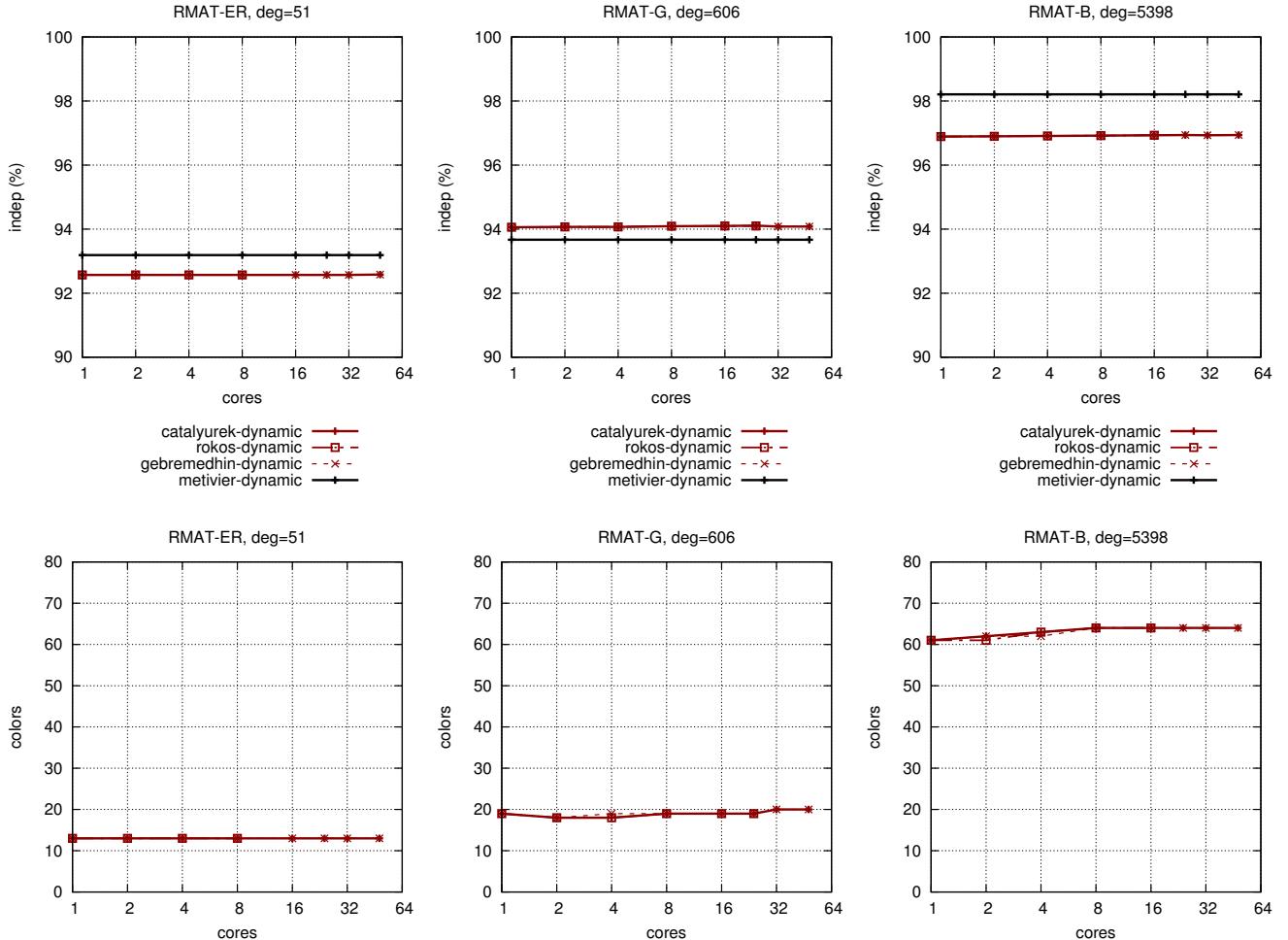


Figure B.6: Ratio de taille des stables  $\mathcal{U}^*$  sur  $n$ , et nombre effectif de couleurs  $|\mathbb{P}|$  pour chaque graphe.

Le ratio de taille du stable est plus élevé pour l’instance la plus irrégulière RMAT-B avec  $|\mathcal{U}_B^*| = 97\%$  comparé à l’instance quasi-régulière RMAT-ER avec  $|\mathcal{U}_{ER}^*| = 92\%$ . Ainsi la qualité de la solution est meilleure sur RMAT-B pour maxindset. A contrario, le nombre de couleurs utilisées est plus élevé pour RMAT-B avec  $|\mathbb{P}_B| = 60$  tandis qu’elle est moindre pour RMAT-ER avec  $|\mathbb{P}_B| = 12$ . Ainsi la qualité de la solution est pire sur RMAT-B pour coloring. Notons toutefois que  $|\mathbb{P}|$  reste négligeable devant  $\Delta + 1$  la borne théorique sur le nombre chromatique de chaque graphe. Cela s’explique par le fait que pour un sommet  $v$  on choisit toujours la plus petite couleur  $c = \max_{u \in \mathcal{N}[v]} \text{color}[u] + 1$  non attribuée à un voisin de  $v$  : quand le voisinage de  $v$  est grand alors  $c$  ainsi que  $|\mathbb{P}|$  sont également grands. Notons également que le stable de cardinalité maximale coïncide toujours avec le stable de plus petite couleur  $\mathcal{U}_1$  de  $\mathbb{P}$  puisque les couleurs sont attribuées par ordre croissant. Ainsi quand le degré du graphe est élevé, la probabilité d’avoir un nombre important de sommets colorés par la plus petite couleur est élevée, ce qui peut expliquer les résultats obtenus sur RMAT-B pour maxindset.

Enfin on constate que les trois heuristiques spéculatives produisent les mêmes résultats puisqu’elles sont basées sur le même noyau séquentiel First-fit, tandis que l’approche Monte-Carlo est légèrement meilleure pour maxindset. En effet cette dernière tient compte du degré du sommet  $v$ , contrairement aux méthodes spéculatives, puisqu’il est rajouté dans  $\mathcal{U}^*$  avec une probabilité de  $1/2\deg[v]$ . Ainsi il délaisse les sommets à fort degré au profit de ceux à faible degré mais beaucoup plus nombreux, ce qui résulte en un stable de cardinalité plus élevée, notamment pour RMAT-B.

### B.1.4 Synthèse

CONSTATS. L'étude expérimentale présentée dans la section B.1.3 a révélé un certain nombre de constats sur les approches spéculatives, notamment :

- ✓ leur efficacité : elles sont 100 fois plus rapides qu'une approche Monte-Carlo (voir figure B.3), pour une solution approximative  $\mathcal{U}^*$  de qualité quasi-identique (de l'ordre de 90% sur RMAT, voir figure B.6). De plus les solutions extraites  $\mathcal{U}^*$  et  $\mathbb{P}$  sont similaires à celles obtenues par le noyau séquentiel. Par ailleurs, les ratios de conflits ne représentent même pas 1% de  $n$  pour chaque instance de RMAT mais ont tendance à croître proportionnellement avec la granularité des tâches (voir figure B.5). Près de 80% des conflits ont lieu au premier itéré, ce qui résulte en un faible nombre d'itérés pour les deux approches itératives décrites dans [204, 206].
- ✗ leur sensibilité aux politiques d'ordonnancement et de placement mémoire : le déséquilibre de charges est le principal facteur limitatif à leur scalabilité. Elles passent mieux à l'échelle quand la granularité des tâches est suffisamment fine, typiquement de l'ordre 0.002% sur RMAT pour  $n = O(10^6)$  (voir figure B.3), néanmoins elle ne doit pas être trop fine à nombre  $p$  de cores élevé. De plus, l'accélération peut être significativement mitigée à  $p$  élevé si le placement mémoire n'est pas correctement géré (voir figure B.4) notamment en contexte NUMA. L'inconvénient est ce que cela réduit la portabilité de leurs performances.

Partant de ces constats, nous décidons de réutiliser certaines briques en vue de construire notre propre heuristique pour `maxindset`. Plus précisément, nous dérivons notre heuristique à partir de celle de Catalyurek notamment parce que celle de Gebremedhin comporte une étape séquentielle dans la phase de correction, et que celle de Rokos n'a même pas pu se terminer en raison des effets SIMD. Cette non terminaison est induite par le fait de recolorer immédiatement chaque sommet défectueux.

## B.2 EXTRACTION D'UN COUPLAGE MAXIMAL

### B.2.1 Cadre et motivations

Notons que notre stratégie basée sur la récupération d'un stable  $\mathcal{U}^*$  pour l'extraction de tâches non conflictuelles est suffisamment générique pour être appliquée à l'ensemble des noyaux. Pour chaque noyau, il implique juste la formulation des dépendances de données en un graphe adéquat. Pour le noyau de relaxation en particulier, nous avons besoin d'identifier un ensemble d'arêtes disjointes à ordonner, noté  $S$  puisqu'ici chaque tâche élémentaire correspond à la bascule d'une arête.

Dans une quête perpétuelle de localité, nous avions initialement recours à une carte combinatoire pour la représentation de la topologie de la triangulation. Il s'agit d'une structure de données topologique dans laquelle les relations d'adjacence des arêtes de la triangulation étaient exprimées explicitement. Ainsi lors d'une bascule d'une arête  $e$ , seules les demi-arêtes relatives aux deux mailles incidentes  $K$  et  $R$  étaient impliquées dans la mise à jour des données topologiques. Naturellement, notre idée était de construire un graphe  $G$  où chaque sommet correspond à une arête de la triangulation, et chaque arête exprime le fait que deux arêtes de la triangulation sont adjacentes comme illustré sur la figure B.7. Ainsi pour obtenir  $S$ , il nous suffisait d'extraire un stable de cardinalité maximale de  $G$  et donc de réutiliser la même heuristique définie à la section B.1.4 pour ce noyau.

MOTIVATIONS. Pour le noyau de relaxation, le ratio de tâches parallèles obtenu ainsi que son évolution au fur et à mesure des itérés était similaire au cas de la contraction car le graphe associé était également planaire avec un degré fixe  $\Delta = 4$  comme illustré sur la figure B.7. Néanmoins l'accélération des autres noyaux était mitigée par le nombre important de défaut de caches induites par les requêtes de voisinage à la carte combinatoire, notamment pour la contraction (voir figure B.91). Ainsi nous avions dû changer de structure de données afin de limiter le ratio d'indirections lors des requêtes topologiques. Pour cela, nous avons opté pour une structure de données **centrée-nœud** dans laquelle les références aux mailles incidentes à chaque point sont stockées explicitement mais pas les arêtes. En termes d'accès-mémoire, cela nous permet de reconstruire rapidement la boule d'un point car il y a moins d'indirections comme indiqué sur la figure B.9. Toutefois cela soulève un inconvénient de

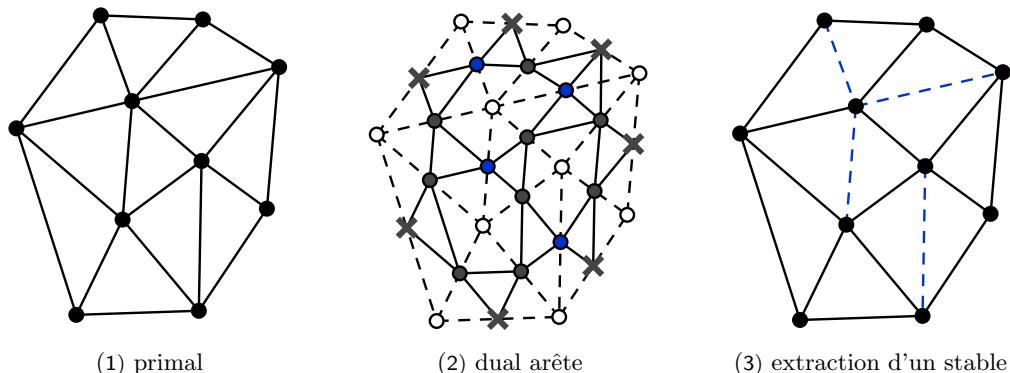


Figure B.7: Graphe de tâches initialement utilisé pour le noyau de relaxation. Ici les sommets du graphe correspondent aux arêtes de la triangulation, et les arêtes du graphe désignent leur relations d'incidence. Ainsi extraire un stable du graphe revient à identifier les arêtes non conflictuelles de la triangulation.

taille : la construction de  $G$  devenait trop coûteuse pour le noyau de relaxation car on a plus une représentation explicite des arêtes de la triangulation, ainsi que leur relations d'adjacence. Cela est d'autant plus grave dans la mesure où l'extraction de  $S$  s'effectuait à chaque itéré du noyau. Ainsi la solution initiale n'était plus viable. En recherche d'une autre alternative, nous décidons de traiter plutôt les paires de mailles au lieu des arêtes. Dans ce cadre, notre idée est de construire  $G$  à partir du graphe dual de la triangulation illustré à la figure 4.6. Ainsi trouver un couplage des sommets de  $G$  revient à trouver un ensemble disjoint de paire de mailles à traiter.

### B.2.2 Cas d'un graphe non-biparti.

**Définition 31** (CHEMINS AUGMENTANTS). Soit  $G = (V, E)$  un graphe non orienté et  $\mathcal{U}$  un couplage de  $G$ . Un sommet est dit libre s'il ne touche aucune arête de  $\mathcal{U}$ . Un chemin de  $G$  est dit alternant si ses arêtes sont alternativement dans  $\mathcal{U}$  et hors de  $\mathcal{U}$  (donc si une arête sur deux est dans  $\mathcal{U}$ ). Un chemin augmentant est un chemin alternant dont les points de départ et d'arrivée sont libres.

**TRAVAUX CONNEXES.** Rappelons qu'en théorie de graphes, un couplage est un appariement unique des sommets d'un graphe  $G = (V, E)$ . Ainsi calculer un couplage de  $G$  revient à extraire un ensemble d'arêtes  $\mathcal{U} \subset E$  deux à deux disjointes, ce qui est équivalent à trouver un stable du graphe dual formé par les arêtes de  $G$  comme sur l'exemple de la figure B.7. Il s'agit d'un problème largement étudié dont une synthèse des noyaux séquentiels est donnée en [195], et celle des approches parallèles résumées à la table B.4. De manière synthétique, il se décline en quatre variantes selon que  $G$  est biparti ou non, et que l'on cherche à maximiser la *cardinalité* ou plutôt les *poids* des arêtes de  $\mathcal{U}$ . Notre cas est restreint à celui d'un couplage de *cardinalité maximale* d'un graphe planaire de degré borné  $\Delta = 3$  mais *non biparti*. Dans le cas des graphes généraux, on peut distinguer :

- l'approche gloutonne (notée *greedy*) qui consiste en un parcours de  $G$  en choisissant de manière arbitraire, ou selon un critère spécifique basé sur les degrés, une arête à rajouter dans  $\mathcal{U}$  tout en la retirant de  $G$ . Les variantes se distinguent sur la manière de choisir l'arête en question. Parmi celles-ci, on distingue notamment celle de Karp-Sipser [196] qui distingue le cas des sommets de degré 1 auquel cas l'arête est automatiquement ajoutée dans  $\mathcal{U}$ , des autres sommets auquel cas l'arête est rajoutée selon une probabilité. Cette approche fournit un couplage pas forcément optimal mais pouvant servir d'initialisation aux algorithmes de recherche locale, y compris pour les graphes bipartis [183].
  - l'approche *Blossom* initié par Edmond en 1983 [197] et ayant donné lieu à plusieurs variantes. Il s'agit d'un algorithme de recherche locale basé sur l'extraction de chemins augmentants de  $G$ .

Listing B.1: par une carte combinatoire

```

1 // reference du brin de départ
2 const int init = mesh.node[v].dart;
3 int curr = init;
4
5 // parcours circulaire de cur->init
6 do {
7     const dart& edge = mesh.darts[curr];
8     const dart& twin = mesh.darts[edge.twin];
9     curr = twin.next;
10    // stocker le sommet pointe par 'edge'
11    neigh.push_back(edge.dest);
12 }
13 while(curr != init);
14 return neigh;

```

Listing B.2: par un graphe d'incidence

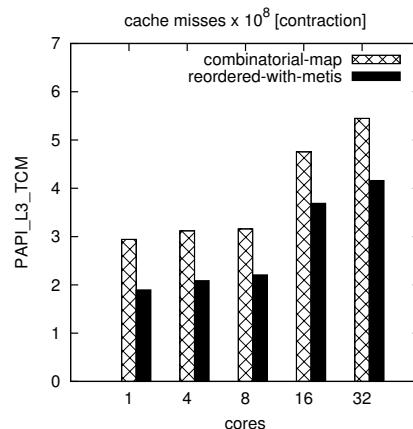
```

1 int i;
2 // parcours des mailles incidentes à v
3 for(const int& k : mesh.stenc[v]){
4     const int* p = mesh.elem[k];
5     // précharger la prochaine maille
6     __builtin_prefetch(&(mesh.elem[k+1]), 0, 1);
7     i = 0;
8     while(p[i] != v);
9     neigh.push_back(p[i+1%3]);
10    neigh.push_back(p[i+2%3]);
11 }
12 // plus cache-friendly qu'un tree
13 std::unique(neigh.begin(),neigh.end());
14 return neigh;

```

Figure B.8: Reconstruction du voisinage d'un point selon la structure de données topologique en C++.

En B.1, chaque demi-arête garde une référence de sa jumelle, sa prochaine et ses deux sommets, tandis qu'en B.2, chaque point stocke les références des mailles qui lui sont incidentes. À chaque itéré, la routine implique deux indirections (lignes 7 et 8) en B.1 et une seule en B.2 (ligne 4). Il est impossible d'anticiper le prochain itéré dans le premier cas, tandis que le nombre d'itérés est connu et les indices de `stenc[v]` sont déjà en cache dans le second cas : il est ainsi possible de précharger `elem[k]` pour le prochain `k`.



(1) défaut de cache L3

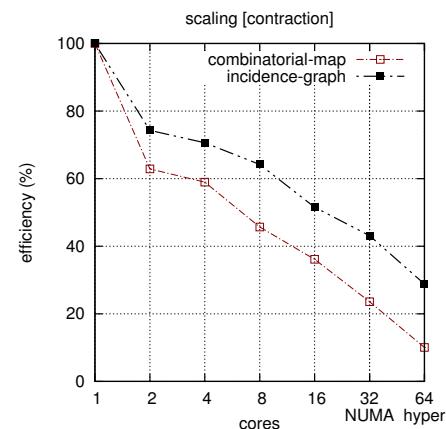
(2) efficacité  $T_1/(pT_p)$  en %

Figure B.9: Impact des indirections mémoire sur le scaling du noyau de contraction sur Haswell. En (1), le recours à la carte combinatoire rend le noyau très sensible au placement mémoire des données topologiques. En (2), la réduction des indirections mémoire par le recours à un graphe d'indication permet d'améliorer significativement l'efficacité du noyau.

au sens de la définition 31, avec évitement de cycles par **contraction** des corolles. En effet un couplage est maximal s'il n'existe pas de chemins augmentants par rapport à lui : ainsi l'idée est de saturer  $\mathcal{U}$  par suppression de chemins augmentants depuis les sommets sources, ce qui se fait en  $O(n^2m)$ . La réelle différence entre les graphes bipartis et généraux réside dans la présence de cycles dans ce dernier cas, comme illustré à la figure B.10. Autant la recherche peut s'effectuer par un simple parcours en profondeur (notée **DFS**) dans le premier cas, autant sa terminaison nécessite un traitement particulier des cycles dans le cas général. Ici l'idée est de contracter les arêtes d'une corolle en un sommet  $c$  et calculer le couplage  $\mathcal{U}'$  du graphe  $G'$  induit par  $V \cup \{c\}$  privé de cette corolle. En effet s'il existe un chemin augmentant pour  $\mathcal{U}'$  dans  $G'$ , alors il en

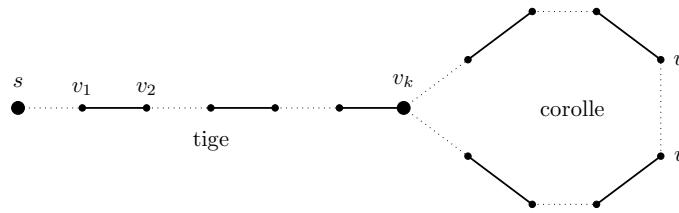


Figure B.10: Présence de cycles lors d'une recherche de chemin augmentant dans un graphe non biparti.

existe de même dans  $G$  d'après le lemme de Berge [198].

Table B.4: Synthèse des algorithmes parallèles d'approximation pour le couplage de graphes.

approche	recherche	granularité	général	architecture	cf.
Locally dominant	locale, synchros par queue	un bloc	✓	multicore	[213]
Suitor	locale, synchros par verrous	un bloc	✓	manycore	[214]
Pothen-Fan	chemins augmentants par DFS	un sommet	✗	multicore	[183]
MS-BFS	chemins augmentants par BFS	un sommet	✗	manycore	[215]
Hopcroft-Karp	mixte BFS et DFS	multiple	✗	hétérogène	[216]
Push-relabel	label guided FIFO search	un sommet	✗	hétérogène	[217]
Karp-Sipser	locale, synchros par atomics	un bloc	✓	manycore	[218]

PARALLÉLISATION. Le couplage est un problème irrégulier à l'instar de l'extraction de stables ou de la coloration [212]. Presque 80% des travaux récents [183, 213–218] se concentrent sur les graphes bipartis aussi bien pour le couplage de cardinalité ou de poids maximal, puisqu'ils sont conceptuellement plus simples à gérer.

- **Blossom :** Bien qu'elle soit l'approche de référence pour les graphes non-bipartis, elle est trop coûteuse pour être applicable dans notre contexte, notamment en raison de la contraction récursive et multi-cas de corolles lors de la recherche de chemins augmentants. De plus réaliser un portage efficient sur des architectures multicore est ardue à cause du surcoût induit par les synchronisations à grain fin, et le peu de tentatives existantes sont inefficentes ou inadaptées à notre contexte : celle décrite dans [202] nécessite une dizaine de secondes pour traiter  $n = o(200)$  sommets, et aucune accélération substantielle n'est obtenue pour un graphe épars (pas d'étude de **strong scaling**), tandis que celle implémentée dans [203] implique des synchronisations à base de verrous, ce qui est incompatible avec notre approche lock-free.
- **Monte-Carlo :** La randomisation de l'approche **greedy** fournit une implémentation parallèle directe néanmoins nous sommes confrontés au même problème de surcoût relatif à la génération des poids ou graines aléatoires évoqué au paragraphe B.1.2. On peut noter toutefois l'approche originale décrite dans [223] dédiée au portage de **greedy** sur GPU, initialement conçue pour accélérer la phase de contraction lors d'un partitionnement de graphes multi-niveaux. Elle est basée sur l'assignation de couleurs aux sommets de  $G$  qui peuvent être rouges ou bleus selon une loi de Bernoulli : les sommets bleus proposent à leurs voisins rouges qui leur répondent de manière randomisée selon la même loi de probabilité. Ainsi si les réponses sont compatibles alors l'arête est rajoutée dans  $\mathcal{U}$ . L'algorithme affiche un très bon **strong scaling** néanmoins le noyau de sélection de couleur invoqué à chaque itéré est trop coûteux au vu de nos contraintes.

### B.2.3 Synthèse

En recherche d'alternatives viables, nous nous sommes intéressés aux stratégies inhérentes aux approches dédiées aux graphes bipartis et basées sur la recherche de chemins augmentants [183, 215,

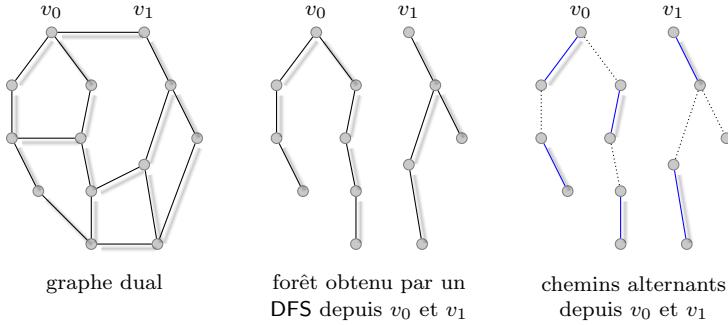


Figure B.11: Extraction du couplage par recherche multi-source de chemins alternants.

[216]. De manière synthétique, elles sont basées sur une extraction multi-source de chemins augmentants par le biais d'une recherche en largeur (BFS) et/ou en profondeur d'abord (DFS) ainsi qu'au recours aux synchronisations à granularité fine (verrous, queue-based etc.).

Table B.5: Comparaison des approches basées sur la recherche de chemins augmentants.

critères	DFS-based [183, 216, 218]	BFS-based [215, 216]
longueur chemins	assez long	court
parallélisation	par sommet source	par niveau
synchronisation	à chaque itéré (grain moyen)	à chaque niveau (grain fin)
facteur limitatif	déséquilibre de charges	surcoût des synchronisations
irrégularité	plus sensible à l'ordonnancement	moins sensible à l'ordonnancement
convergence	plus d'itérés peu coûteux	peu d'itérés plus coûteux
scaling	bonne strong scaling	$t_{\max}$ invariant; bonne weak scaling.
architecture	manycore	massivement multithread (GPGPU)

Pour un portage efficient, les auteurs préconisent une approche de conception codesign tirant profit des spécificités des architectures cibles (version **dataflow** et recours aux primitives natives pour la synchronisation), en raison de la forte irrégularité du problème (voir [212] pour plus de détails). Sans rentrer dans leurs spécificités, nous en avons identifié et résumé les points-clés d'un portage efficient de ces approches à la table B.5 : cela nous permettra d'identifier les adaptations nécessaires au design de notre propre algorithme.

\*\*\*

## RÉFÉRENCES

### Maillages et tessellations 3D

#### Livres, thèses et rapports techniques

- [1] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Levy. *Polygon Mesh Processing*. AK Peters, 2010. ISBN: 978-1-56881-426-1.
- [2] Franck Ledoux. “Maillage hexaédrique pour la simulation numérique : représentations et algorithmes”. Habilitation thesis. Université de Poitiers, 2014.
- [3] Paul-Louis George and Houman Bourouchaki. *Delaunay triangulation and meshing: Application to finite elements*. Hermès, 1998.
- [4] Jonathan Shewchuk. *Lecture notes on Delaunay mesh generation*. Tech. rep. 2012.
- [5] Elefterios Melissaratos. *L<sup>p</sup> optimal d-dimensional triangulations for piecewise linear interpolation: A new result on data dependent triangulations*. Tech. rep. RUU-CS-93-13. Utrecht University, 1993.
- [6] Charles Lawson. *Generation of a Triangular Grid with Applications to Contour Plotting*. Tech. rep. 299. California Institute of Technology, 1977.
- [7] Robert Pless. *Lecture 17: Voronoi Diagrams and Delaunay Triangulations*. Tech. rep. 2003. URL: <https://research.engineering.wustl.edu/~pless/506/l17.html>.
- [8] Paul Chew. *Guaranteed-Quality Triangular Meshes*. Tech. rep. 89-983. Cornell University, 1989.
- [9] The CGAL Project. *CGAL User and Reference Manual*. 4.11. CGAL Editorial Board, 2017. URL: <http://doc.cgal.org/4.11/Manual/packages.html>.
- [10] Pascal Frey and Houman Borouchaki. *Texel: triangulation de surfaces implicites. Partie I: aspects théoriques*. Research Report RR-3066. INRIA, 1996.
- [11] Pierre Alliez, Giuliana Ucelli, Craig Gotsman, and Marco Attene. “Recent Advances in Remeshing of Surfaces”. In: *Shape Analysis and Structuring*. Ed. by Leila De Floriani and Michela Spagnuolo. Springer Berlin Heidelberg, 2008, pp. 53–82. ISBN: 978-3-540-33265-7.
- [12] Pascal Frey and Paul-Louis George. *Maillages: applications aux éléments finis*. Hermès Science Publications, 1999. ISBN: 9782746200241.
- [13] Nikos Chrisochoides. *A Survey of Parallel Mesh Generation Methods*. Tech. rep. Brown University, Providence, 2005.

#### Articles et actes de conférences

- [14] Samuel Rippa. “Minimal roughness of a Delaunay triangulation”. *Computer Aided Geometric Design* 7 (1990), pp. 489–497.
- [15] Samuel Rippa. “Long and thin triangles can be good for linear interpolation”. *SIAM Journal on Numerical Analysis* 29 (1992), pp. 257–270.
- [16] Jonathan Shewchuk. “Unstructured mesh generation”. *Combinatorial Scientific Computing* (2012), pp. 257–297.
- [17] Adrian Bowyer. “Computing Dirichlet Tessellations”. In: *Computing Journal*. Vol. 24. 1981, pp. 162–166.
- [18] David Watson. “Computing The n-dimensional Delaunay Tessellation With Application to Voronoi Polytopes”. In: *Computing Journal*. Vol. 24. 1981, pp. 167–172.
- [19] Jonathan Shewchuk. “Robust adaptive floating-point geometric predicates”. In: *Proceedings of the 12th Annual Symposium on Computational Geometry*. 1996, pp. 141–150.
- [20] Kevin Brown. “Voronoi diagrams from convex hulls”. *Information Processing Letters* 9 (1979), pp. 223–228.

- [21] Herbert Edelsbrunner and Raimund Seidel. “Voronoi diagrams and arrangements”. *Discrete & Computational Geometry* 1 (1986), pp. 25–44.
- [22] Guy Blelloch, Gary Miller, and Dafna Talmor. “Developing a practical projection based parallel delaunay triangulation”. In: *Proceedings of the 12th annual symposium on computationnal geometry*. 1996, pp. 61–83.
- [23] Bradford Barber, David Dobkin, and Hannu Huhdanpaa. “The Quickhull Algorithm for Convex Hulls”. *ACM Trans. Math. Softw.* 22 (1996), pp. 469–483.
- [24] Michael Murphy, David Mount, and Carl Gable. “A point-placement strategy for conforming Delaunay tetrahedralization”. In: *Proceedings of the 11th ACM-SIAM symposium on Discrete algorithms*. 2000, pp. 67–74.
- [25] David Cohen-Steiner, Éric Colin-de-Verdière, and Mariette Yvinec. “Conforming Delaunay triangulation in 3D”. In: *Proceedings of the 18th annual ACM symposium on computational geometry*. 2002.
- [26] Hang Si and Klaus Gärtner. “Meshing Piecewise Linear Complexes by Constrained Delaunay Tetrahedralizations”. In: *Proceedings of the 14th International Meshing Roundtable*. 2005, pp. 147–163.
- [27] Der-Tsai Lee and AK. Lin. “Generalized Delaunay triangulations for planar graphs”. *Discrete and Computational Geometry* 1 (1986), pp. 201–217.
- [28] Edward Verbree and Hang Si. “Validation and Storage of Polyhedra through Constrained Delaunay Tetrahedralization”. *Proceedings of the 5th International Conference on Geographic Information Science* (2008), pp. 354–369.
- [29] Jonathan Shewchuk. “Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator”. *Applied Computational Geometry: Towards Geometric Engineering* 1148 (1996), pp. 203–222.
- [30] Hang Si. “TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator”. *ACM Trans. Math. Softw.* 41 (2015), 11:1–11:36.
- [31] Alexandra Claisse, Vincent Ducrot, and Pascal Frey. “Levelsets and anisotropic mesh adaptation”. *Discrete and Continuous Dynamical Systems* 23 (Sept. 2008), pp. 165–183.
- [32] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. “Mesh Optimization”. In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. ACM, 1993, pp. 19–26.
- [33] Yutaka Ohtake, Alexander Belyaev, and Ilia Bogaevski. “Mesh regularization and adaptive smoothing”. *Computer-Aided Design* 33.11 (2001), pp. 789–800.
- [34] Adrien Loseille. “Metric-orthogonal anisotropic mesh generation”. In: *Proceedings of the 23rd International Meshing Roundtable*. 2014, pp. 403–415.

## Extraction d'isosurfaces et visualisation

### Livres, thèses et rapports techniques

- [35] Cyril Crassin. *OpenGL Geometry Shader Marching Cubes*. 2007. URL: [http://www.icare3d.org/codes-and-projects/codes/opengl\\_geometry\\_shader\\_marching\\_cubes.html](http://www.icare3d.org/codes-and-projects/codes/opengl_geometry_shader_marching_cubes.html).
- [36] Christopher Dyken, Gernot Ziegler, Christian Theobalt, and Hans-Peter Seidel. *HPMC: High-speed Marching Cubes using Histogram Pyramids*. 2007. URL: <http://sintefmath.github.io/hpmc/>.
- [37] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit: An Object-oriented Approach to 3D Graphics*. Kitware, 2006. ISBN: 9781930934191.
- [38] Nvidia. URL: <https://developer.nvidia.com/cuda-toolkit>.
- [39] Matt Webcorner. *Marching Cubes*. 2014. URL: <https://graphics.stanford.edu/~mdfisher/MarchingCubes.html>.
- [40] Nvidia. *La tessellation DirectX 11*. URL: [http://www.nvidia.fr/object/tessellation\\_fr.html](http://www.nvidia.fr/object/tessellation_fr.html).

## Articles et actes de conférences

- [41] Eugene Allgower and Phillip Schmidt. "An Algorithm for Piecewise-Linear Approximation of an Implicitly Defined Manifold". *SIAM Journal on Numerical Analysis* 22.2 (1985), pp. 322–346.
- [42] Bruno Rodriguez De-Araújo, Daniel Lopes, Pauline Jepp, Joaquim Jorge, and Brian Wyvill. "A Survey on Implicit Surface Polygonization". *Computing Survey* 47 (2015), pp. 1–39.
- [43] Timothy Newman and Hong Yi. "A survey of the marching cubes algorithm". *Computers & Graphics* 30.5 (2006), pp. 854–879.
- [44] William Lorensen and Harvey Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. ACM, 1987, pp. 163–169.
- [45] Gregory Nielson and Bernd Hamann. "The Asymptotic decider: resolving the ambiguity in Marching cubes". In: *Proceedings of the 2nd Conference in Visualization*. IEEE Computer Society Press, 1991, pp. 83–91.
- [46] Sergey Matveyev. "Approximation of Isosurface in the Marching Cube: Ambiguity Problem". In: *Proceedings of the Conference on Visualization*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994, pp. 288–292.
- [47] Jules Bloomenthal. "Polygonization of Implicit Surfaces". *Comput. Aided Geom. Des.* 5 (1988), pp. 341–355.
- [48] Akoi Doi and Akoi Koide. "An Efficient Method of Triangulating Equi-Valued Surfaces by Using Tetrahedral Cells". *IEICE Transactions on Information and Systems* E74 (1992), pp. 214–224.
- [49] François Labelle and Jonathan Shewchuk. "Isosurface Stuffing: Fast Tetrahedral Meshes with Good Dihedral Angles". *ACM Transactions on Graphics* 26 (2007). Special issue on Proceedings of SIGGRAPH 2007, pp. 57.1–57.10.
- [50] Timothy Newman and Hong Yi. "A survey of the marching cubes algorithm". *Computers and Graphics* 30.5 (2006), pp. 854–879.
- [51] Allen van Gelder and Jane Wilhelms. "Topological Considerations in Isosurface Generation". *ACM Transactions on Graphics* 13 (1994), pp. 337–375.
- [52] Philip Sutton and Charles Hansen. "Accelerated Isosurface Extraction in Time-Varying Fields". *IEEE Transactions on Visualization and Computer Graphics* 6 (2000), pp. 98–107.
- [53] Yi-Jen Chiang. "Out-of-core isosurface extraction of time-varying fields over irregular grids". In: *IEEE Visualization*. 2003, pp. 217–224.
- [54] Chandrajit Bajaj, Valerio Pascucci, and Daniel Schikore. "Fast Isocontouring for Improved Interactivity". In: *Proceedings of 1996 Symposium on Volume Visualization*. 1996, pp. 39–46.
- [55] Christopher Dyken, Gernot Ziegler, Christian Theobalt, and Hans-Peter Seidel. "High-speed Marching Cubes using HistoPyramids". *Computer Graphics Forum* (2008).
- [56] Erik Smistad, Mohammadmehd Bozorgi, and Frank Lindseth. "FAST: framework for heterogeneous medical image computing and visualization". *International Journal of Computer Assisted Radiology and Surgery* 10 (2015), pp. 1811–1822.
- [57] Sung-Eui Yoon and Peter Lindstrom. "Mesh Layouts for Block-Based Caches". *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 1213–1220.
- [58] Marc Tchiboukdjian, Vincent Danjean, and Bruno Raffin. "Binary Mesh Partitioning for Cache-Efficient Visualization". *IEEE Transactions on Visualization and Computer Graphics* 16.5 (2010), pp. 815–828.

## CFD, remaillage variationnel et anisotrope

### Livres, thèses et rapports techniques

- [59] Adrien Loseille. “Anisotropic 3D hessian-based multi-scale and adjoint-based mesh adaptation for CFD. Application to high fidelity sonic boom prediction”. PhD thesis. Université Pierre et Marie Curie, Paris, 2008.
- [60] Frédéric Alauzet. *Metric-based anisotropic mesh adaptation*. Summer school on numerical analysis CEA-EDF-INRIA. INRIA, 2010.
- [61] Cécile Dobrzynski. “Adaptation de maillage anisotrope 3D et application l'aéro-thermique des bâtiments”. PhD thesis. Université Pierre et Marie Curie, 2005.
- [62] Max Wardetzky. “Convergence of the Cotangent Formula: An Overview”. In: *Discrete Differential Geometry*. Birkhäuser Basel, 2008, pp. 275–286. ISBN: 978-3-7643-8621-4.
- [63] André Bakker. *Meshing lecture : Applied Computational Fluid Dynamics*. 2006. URL: <http://www.bakker.org/dartmouth06/engs150/07-mesh.pdf>.
- [64] Patrick Knupp, David Thompson, Corey Ernst, David Stimpson, and Philippe Pebay. *The verdict geometric quality library*. Tech. rep. SAND2007-1751. Sandia National Laboratories, USA, 2006.
- [65] A.M. Winslow. *Adaptive-mesh Zoning by the Equipotential Method*. Tech. rep. Lawrence Livermore National Laboratory, 1981.
- [66] Weizhang Huang and Robert Russell. *Adaptive Moving Mesh Methods*. Springer, 2011. ISBN: 9781441979155.

### Articles et actes de conférences

- [67] Pierre Sagaut, Sebastian Deck, and Marc Terracol. “Multiscale and multiresolution approaches in turbulence”. *Journal of Fluid Mechanics* 646 (2006), pp. 537–538.
- [68] Jonathan Shewchuk. “What Is a Good Linear Finite Element? - Interpolation, Conditioning, Anisotropy, and Quality Measures”. In: *Proceedings of the 11th International Meshing Roundtable*. Vol. 73. Sept. 2002.
- [69] Weiming Cao. “On the Error of Linear Interpolation and the Orientation, Aspect Ratio, and Internal Angles of a Triangle”. *SIAM Journal on Numerical Analysis* 43.1 (2005), pp. 19–40.
- [70] Philippe Pébay and Timothy Baker. “Analysis of Triangle Quality Measures”. *Math. Comput.* 72.244 (2003), pp. 1817–1839.
- [71] Patrick Knupp. “Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities”. *Journal of Numerical Methods in Engineering* 48 (2000), pp. 1165–1185.
- [72] Jan Brandts, Antti Hannukainen, Sergey Korotov, and Michal Krizek. “On angle conditions in the finite element method”. *SeMA Journal* 56 (2011), pp. 81–95.
- [73] Patrick Knupp. “Algebraic mesh quality metrics”. *SIAM Journal of Scientific Computing* 23 (2001), pp. 193–218.
- [74] Weizhang Huang. “Metric Tensors for Anisotropic Mesh Generation”. *Journal of Computational Physics* 204 (2005), pp. 633–665.
- [75] Frédéric Alauzet, Adrien Loseille, Alain Dervieux, and Pascal Frey. “Multi-Dimensional Continuous Metric for Mesh Adaptation”. In: *Proceedings of the 15th International Meshing Roundtable*. Berlin, 2006.
- [76] PW. Power, Chris Pain, Matthew Piggott, Fangxin Fang, Gerard Gorman, Adrian Umpleby, Anthony Goddard, and Michael Navon. “Adjoint A Posteriori Error Measures for Anisotropic Mesh Optimisation”. *Computers & Mathematics with Applications* 52.8 (2006), pp. 1213–1242.
- [77] Hehu Xie and Xiaobo Yin. “Metric tensors for the interpolation error and its gradient in  $L^p$  norm”. *Journal of Computational Physics* 256 (2014), pp. 543–562.

- [78] Frédéric Alauzet and Adrien Loseille. “A Decade of Progress on Anisotropic Mesh Adaptation for Computational Fluid Dynamics”. *Computer Aided Design* 72 (2016), pp. 13–39.
- [79] Jeremiah Brackbill and Jeffrey Saltzman. “Adaptive zoning for singular problems in two dimensions”. *Journal of Computational Physics* 46.3 (1982), pp. 342–368.
- [80] A. Dvinsky. “Adaptive Grid Generation from Harmonic Maps on Riemannian Manifolds”. *Journal of Computational Physics* 95 (1991), pp. 450–476.
- [81] Weizhang Huang. “Variational Mesh Adaptation: Isotropy and Equidistribution”. *Journal of Computational Physics* 174.2 (2001), pp. 903–924.
- [82] Qiang Du and Desheng Wang. “Anisotropic Centroidal Voronoi Tessellations and Their Applications”. *SIAM Journal on Scientific Computing* 26.3 (2005), pp. 737–761.
- [83] Sébastien Valette and Jean-Marc Chassery. “Approximated Centroidal Voronoi Diagrams for Uniform Polygonal Mesh Coarsening”. *Computer Graphics Forum* 23.3 (2004). Eurographics 2004 proceedings, pp. 381–389. URL: <https://hal.archives-ouvertes.fr/hal-00534535>.
- [84] Pierre Alliez, Éric Colin de Verdière, Olivier Devillers, and Martin Isenburg. “Centroidal Voronoi Diagrams for Isotropic Surface Remeshing”. *Graph. Models* 67.3 (2005), pp. 204–231.
- [85] Sébastien Valette, Jean-Marc Chassery, and Rémy Prost. “Generic remeshing of 3D triangular meshes with metric-dependent discrete Voronoi Diagrams”. *IEEE Transactions on Visualization and Computer Graphics* 14.2 (2008), pp. 369–381.
- [86] Dong-Ming Yan and Peter Wonka. “Non-Obtuse Remeshing with Centroidal Voronoi Tessellation”. *IEEE Transactions on Visualization & Computer Graphics* 22.9 (2016), pp. 2136–2144.
- [87] Jeanne Pellerin, Bruno Lévy, Guillaume Caumon, and Arnaud Botella. “Automatic surface remeshing of 3D structural models at specified resolution: A method based on Voronoi diagrams”. *Computers and Geosciences* 62 (2016), pp. 103–116.
- [88] Bruno Lévy and Nicolas Bonneel. “Variational Anisotropic Surface Meshing with Voronoi Parallel Linear Enumeration”. In: *Proceedings of the 21st International Meshing Roundtable*. 2012.
- [89] Sébastien Valette, Julien Dardenne, Nicolas Siauve, and Rémy Prost. “Génération de Mailles Triangulaires Adaptatifs 2D avec des Diagrammes de Voronoi Centroïdaux”. In: *Colloque GRETSI – Traitement du Signal et des Images*. 22. 2009.
- [90] Lloyd. “Least squares quantization in PCM”. *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137.
- [91] Yang Liu, Wenping Wang, Bruno Lévy, Feng Sun, Dong-Ming Yan, Lin Lu, and Chenglei Yang. “On Centroidal Voronoi Tessellation: Energy Smoothness and Fast Computation”. *ACM Trans. Graph.* 28.4 (2009), 101:1–101:17.
- [92] DC. Liu and Jorge Nocedal. “On the Limited Memory BFGS Method for Large Scale Optimization”. *Math. Program.* 45.3 (1989), pp. 503–528.
- [93] Julien Dompierre, Yvan Mokwinski, Marie-Gabrielle Vallet, and François Guibault. “On ellipse intersection and union with application to anisotropic mesh adaptation”. *Engineering with Computers* 33.4 (2017), pp. 745–766.
- [94] VincentArsigny, Pierre Fillard, Xavier Pennec, and Nicholas Ayache. “Fast and Simple Calculus on Tensors in the Log-Euclidean Framework”. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2005: 8th International Conference*. Springer Berlin Heidelberg, 2005, pp. 115–122.
- [95] Cécile Dobrzynski and Pascal Frey. “Anisotropic Delaunay mesh adaptation for unsteady simulations”. In: *Proceedings of the 17th International Meshing Roundtable*. 2008, pp. 177–194.
- [96] Gaëtan Compère, Jean-François Remacle, Johan Jansson, and Johan Hoffman. “A mesh adaptation framework for dealing with large deforming meshes”. *International Journal for Numerical Methods in Engineering* 82.7 (2010), pp. 843–867.
- [97] Adrien Loseille and Victorien Menier. “Serial and Parallel Mesh Modification Through a Unique Cavity-Based Primitive”. In: *23rd International Meshing Roundtable*. 2014, pp. 541–558.

- [98] François Courty, David Leservoisier, Paul-Louis George, and Alain Dervieux. “Continuous metrics and mesh adaptation”. *Applied Numerical Mathematics* 56.2 (2006), pp. 117–145.
- [99] Xiangrong Li, Jean-François Remacle, and Nicolas Chevaugeon. “Anisotropic mesh gradation control”. In: *Proceedings of the 13th International Meshing Roundtable*. 2004.
- [100] Houman Borouchaki, Frédéric Hecht, and Pascal Frey. “Mesh gradation control”. *International Journal for Numerical Methods in Engineering* 43.6 (1998), pp. 1143–1165.
- [101] Frédéric Alauzet. “Size gradation control of anisotropic meshes”. *Finite Elements in Analysis and Design* 46.1 (2010), pp. 181–202.
- [102] Adrien Loseille, Victorien Menier, and Frédéric Alauzet. “Parallel Generation of Large-Size Adapted Meshes”. In: *24<sup>th</sup> International Meshing Roundtable*. 2015, pp. 57–69.
- [103] Marie-Gabrielle Vallet, CM. Manole, Julien Domptierre, Steven Dufour, and François Guibault. “Numerical comparison of some Hessian recovery techniques”. In: *International Journal for Numerical Methods in Engineering*. Vol. 72. 2007, pp. 987–1007.

## Reconstruction, discréétisation et optimisation de surfaces

### Livres, thèses et rapports techniques

- [104] Cédric Gérot. *Paramétrisation de maillages*. URL: [http://www.gipsa-lab.grenoble-inp.fr/~cedric/gerot/mes\\_documents/parametrisation.pdf](http://www.gipsa-lab.grenoble-inp.fr/~cedric/gerot/mes_documents/parametrisation.pdf).
- [105] NeoMansLand. *La carte de France par anamorphose de la SNCF*. URL: <http://neomansland.over-blog.org/article-20926040.html>.
- [106] BBC Earth. *These are the discoveries that made Stephen Hawking famous*. URL: <http://www.bbc.com/earth/story/20160107-these-are-the-discoveries-that-made-stephen-hawking-famous>.
- [107] Pascal Frey. *About Surface Remeshing*. Tech. rep. INRIA Gamma Project, 2000.
- [108] Pascal Frey. *YAMS A fully Automatic Adaptive Isotropic Surface Remeshing Procedure*. Tech. rep. RT-0252. INRIA, 2001, p. 36.
- [109] Guillaume Damiand and Pascal Lienhardt. *Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing*. 1st ed. A.K. Peters, Ltd., 2014.
- [110] Alex Yvert. “Modélisation hiérarchique de surfaces à partir de maillages polyédriques et applications”. PhD thesis. INP Grenoble, 2004.
- [111] Yingbin Liu. “Triangular Bézier Surfaces with Approximate Continuity”. PhD thesis. University of Waterloo, 2008.
- [112] Charles Loop. “Smooth Subdivision Surfaces Based on Triangles”. MA thesis. The University of Utah, 1987.
- [113] Daniel Aubram. *Optimization-based smoothing algorithm for triangle meshes over arbitrarily shaped domains*. Tech. rep. 2014. URL: <https://arxiv.org/abs/1410.5977v1>.
- [114] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. *Metro: Measuring Error on Simplified Surfaces*. Tech. rep. Paris, 1996.

### Articles et actes de conférences

- [115] Michael Garland and Paul Heckbert. “Surface Simplification Using Quadric Error Metrics”. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. 1997, pp. 209–216.
- [116] Gabriel Taubin. “Curve and Surface Smoothing Without Shrinkage”. In: *Proceedings of the 5th International Conference on Computer Vision*. ICCV '95. 1995, pp. 852–857.
- [117] Joerg Vollmer, Robert Mencl, and Heinrich Müller. “Improved Laplacian Smoothing of Noisy Surface Meshes”. *EuroGraphics '99* (1999), pp. 131–138.

- [118] Panagiotis Foteinos and Nikos Chrisochoides. "High Quality Real-time Image-to-mesh Conversion for Finite Element Simulations". In: *Proceedings of the 27th International Conference on Supercomputing*. ICS '13. ACM, 2013, pp. 233–242.
- [119] Jean-François Remacle, Christophe Geuzaine, Gaëtan Compère, and Emilie Marchandise. "High-quality surface remeshing using harmonic maps". *International Journal for Numerical Methods in Engineering* 83.4 (2010), pp. 403–425.
- [120] Vitaly Surazhsky and Craig Gotsman. "Explicit Surface Remeshing". In: *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*. SGP '03. 2003, pp. 20–30.
- [121] Rao Garimella, Mikhail Shashkov, and Patrick Knupp. "Triangular and Quadrilateral Surface Mesh Quality Optimization using Local Parametrization". *Computer Methods in Applied Mechanics and Engineering* 193.9-11 (2004), pp. 913–928.
- [122] Michael Floater and Kai Hormann. "Surface Parameterization: a Tutorial and Survey". *Advances in Multiresolution for Geometric Modelling* (2005), pp. 157–186.
- [123] Jean-Daniel Boissonnat, Kan-Le Shi, Jane Tournois, and Mariette Yvinec. "Anisotropic Delaunay Meshes of Surfaces". *ACM ToG* 34.2 (2015), p. 10.
- [124] Tamal Dey, Guanglian Li, and T. Ray. "Polygonal Surface Remeshing with Delaunay Refinement". In: *Proceedings of the 14th International Meshing Roundtable*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 343–361.
- [125] Maria-Cecilia Rivara. "Mesh Refinement Processes Based on the Generalized Bisection of Simplices". *SIAM Journal on Numerical Analysis* 21.3 (1984), pp. 604–613.
- [126] Lori Freitag, Mark Jones, and Paul Plassmann. "An Efficient Parallel Algorithm for Mesh Smoothing". In: *International Meshing Roundtable*. 1995, pp. 47–58.
- [127] Mohammed Aiffa and Joseph Flaherty. "A geometrical approach to mesh smoothing". *Computer Methods in Applied Mechanics and Engineering* 192 (2003), pp. 4497–4514.
- [128] Nina Amenta, Marshall Bern, and David Eppstein. "Optimal Point Placement for Mesh Smoothing". *Journal of Algorithms* 30.2 (1999), pp. 302–322.
- [129] Walton and Meek. "A triangular G1 patch from boundary curves". *Computer-Aided Design* 28.2 (1996), pp. 113–123.
- [130] Chang-Ki Lee, Hae-Do Hwang, and Seung-Hyun Yoon. "Bézier Triangles with G2 Continuity across Boundaries". *Symmetry* 8.3 (2016).
- [131] Franco Dassi, Andrea Mola, and Hang Si. "Curvature-adapted Remeshing of CAD Surfaces". In: *Proceedings of the 23rd International Meshing Roundtable*. Vol. 82. 2014, pp. 253–265.
- [132] Charles Dapogny, Cécile Dobrzynski, and Pascal Frey. "Three-dimensional adaptive domain remeshing, implicit domain meshing, and applications to free and moving boundary problems". *Journal of Computational Physics* 262 (2014), pp. 358–378.
- [133] Christophe Geuzaine and Jean-François Remacle. "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities". *International Journal for Numerical Methods in Engineering* 79 (11 2009), pp. 1309–1331.
- [134] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. "MeshLab: an Open-Source Mesh Processing Tool". In: *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008.
- [135] Nima Aghdaii, Hamid Younesy, and Hao Zhang. "5–6–7 Meshes: Remeshing and analysis". *Comput. Graph.* 36 (2012), pp. 1072–1083.
- [136] Vincent Vidal, Guillaume Lavoué, and Florent Dupont. "Low Budget and High Fidelity Relaxed 567-remeshing". *Comput. Graph.* 47 (2015), pp. 16–23.
- [137] Xiangmin Jiao and Michael Heath. "Feature detection for surface meshes". In: *Proceedings of 8th International Conference on Numerical Grid Generation in Computational Field Simulations*. 2002, pp. 705–714.

- [138] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. “Ridge-valley Lines on Meshes via Implicit Surface Fitting”. *SIGGRAPH* 23.3 (2004), pp. 609–612.
- [139] Shin Yoshizawa, Alexander Belyaev, and Hans-Peter Seidel. “Fast and Robust Detection of Crest Lines on Meshes”. In: *Proceedings of the ACM Symposium on Solid and Physical Modeling*. SPM ’05. ACM, 2005, pp. 227–232.
- [140] Mario Botsch, Stefan Steinberg, Stefan Bischoff, and Leif Kobbelt. “OpenMesh: A Generic and Efficient Polygon Mesh Data Structure”. In: *OpenSG Symposium 2002*. 2002.
- [141] Max Nelson. “Weights for Computing Vertex Normals from Facet Normals”. *Journal of Graphics Tools* 4.2 (1999), pp. 1–6.
- [142] Shuangshuang Jin, Robert Lewis, and David West. “A comparison of algorithms for vertex normal computation”. *The Visual Computer* 21.1 (2005), pp. 71–82.
- [143] Alex Vlachos, Jörg Peters, Chas Boyd, and Jason L. Mitchell. “Curved PN Triangles”. In: *Proceedings of the 2001 Symposium on Interactive 3D Graphics*. ACM, 2001, pp. 159–166.
- [144] Piper. “Visually smooth interpolation with triangular bezier patches”. *Geometric Modelling: Algorithms and New Trends* (1987), pp. 221–233.
- [145] Rida Farouki and V.T. Rajan. “Algorithms for polynomials in Bernstein form”. *Computer Aided Geometric Design* 5.1 (1988), pp. 1–26.
- [146] Charles Loop, Scott Schaefer, Tianyun Ni, and Ignacio Castaño. “Approximating Subdivision Surfaces with Gregory Patches for Hardware Tessellation”. *ACM Trans. Graph.* 28.5 (2009), 151:1–151:9.
- [147] Hiroaki Chiayokura and Fumihiko Kimura. “Design of Solids with Free-form Surfaces”. In: *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’83. ACM, 1983, pp. 289–298.
- [148] William Schroeder, Jonathan Zarge, and William Lorensen. “Decimation of Triangle Meshes”. *SIGGRAPH Comput. Graph.* 26.2 (1992), pp. 65–70.
- [149] Hugues Hoppe. “Progressive Meshes”. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’96. ACM, 1996, pp. 99–108.
- [150] Gabriel Taubin. “A Signal Processing Approach to Fair Surface Design”. In: *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’95. ACM, 1995, pp. 351–358.
- [151] Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. “Interactive Multi-resolution Modeling on Arbitrary Meshes”. In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’98. ACM, 1998, pp. 105–114.
- [152] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan Barr. “Implicit fairing of irregular meshes using diffusion and curvature flow”. In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’99. 1999, pp. 317–324.
- [153] Ulrich Clarenz, Udo Diewald, and Martin Rumpf. “Anisotropic geometric diffusion in surface processing”. In: *Proceedings on the conference on Visualization*. 2000, pp. 397–405.
- [154] Yutaka Ohtake, Alexander Belyaev, and Ilia Bogaevski. “Polyhedral Surface Smoothing with Simultaneous Mesh Regularization”. In: *Proceedings of the Geometric Modeling and Processing*. 2000, pp. 229–237.
- [155] Xiangmin Jiao. “Volume and Feature Preservation in Surface Mesh Optimization”. In: *Proceedings of the 15th International Meshing Roundtable*. 2006, pp. 359–373.
- [156] Andrew Kuprat, Ahmed Khamayseh, Denise George, and Levi Larkey. “Volume Conserving Smoothing for Piecewise Linear Curves, Surfaces, and Triple Lines”. *Journal of Computational Physics* 172.1 (2001), pp. 99–118.
- [157] Lori Freitag and Paul Plassmann. “Local optimization-based simplicial mesh untangling and improvement”. *Journal of Numerical Methods Engineering* 9 (2000), pp. 109–125.

- [158] Scott Canann, Joseph Tristano, and Matthew Staten. “An approach to combined laplacian and optimization-based smoothing for triangular, quadrilateral, and quad-dominant meshes”. In: *In Proceedings of the 7th International Meshing Roundtable*. 1998, pp. 479–494.
- [159] Lori Freitag. “On Combining Laplacian And Optimization-Based Mesh Smoothing Techniques”. *AMD Trends in Unstructured Mesh Generation* 220 (1999), pp. 37–43.
- [160] Arkady Kheyfets, Warner Miller, and Gregory Newton. “Schild’s Ladder Parallel Transport Procedure for an Arbitrary Connection”. *International Journal of Theoretical Physics* 39.12 (2000), pp. 2891–2898.
- [161] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan Barr. “Discrete Differential-Geometry Operators for Triangulated 2-Manifolds”. In: *Visualization and Mathematics III*. Springer Berlin Heidelberg, 2003, pp. 35–57.

## Architectures parallèles et applications

### Livres, thèses et rapports techniques

- [162] Éric Chaput. *Exascale needs and challenges for aeronautics industry*. Tech. rep. PRACEDays15, 2015.
- [163] Jack Dongarra. *Freely Available Software for Linear Algebra*. 2016. URL: <http://www.netlib.org/utk/people/JackDongarra/la-sw.html>.
- [164] Keshav Pingali, Milind Kulkarni, Donald Nguyen, Martin Burtscher, Mario Mendez-lojo, Dimitrios Prountzos, Xin Sui, and Zifei Zhong. *Amorphous data-parallelism in irregular algorithms*. Tech. rep. 09-05. University of Texas, 2009.
- [165] John McCalpin. *STREAM: Sustainable Memory Bandwidth in High Performance Computers*. Tech. rep. University of Texas at Austin, 1991. URL: <http://www.cs.virginia.edu/stream/>.

### Articles et actes de conférences

- [166] Gene Amdahl. “Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities”. *AFIPS* (1967), pp. 483–485.
- [167] Georgios Rokos and Gerard Gorman. “PRAgMaTIC – Parallel Anisotropic Adaptive Mesh Toolkit”. In: *Facing the Multicore-Challenge III: Aspects of New Paradigms and Technologies in Parallel Computing*. Berlin, Heidelberg: Springer, 2013, pp. 143–144.
- [168] Jean-Louis Gautier Thierry Roch and Gilles Villard. “Regular Versus Irregular Problems and Algorithms”. *IRREGULAR* (1995), pp. 1–25.
- [169] Nikos Chrisochoides, Andrey Chernikov, Christos Antonopoulos, Filip Blagojevic, and Dimitrios Nikolopoulos. “A Multigrain Delaunay Mesh Generation Method for Multicore SMT-based Architectures”. *Journal of Parallel and Distributed Computing* (2009), pp. 589–600.
- [170] Leslie Valiant. “A bridging-model for multicore computing”. *Computer and System Sciences* 77 (2011), pp. 154–166.
- [171] Jonathan Hill, Bill McColl, Dan Stefanescu, Mark Goudreau, Kevin Lang, Satish Rao, Torsten Suel, Thanasis Tsantilas, and Rob Bisseling. “BSPlib: The BSP Programming Library”. *Parallel Computing* 24.14 (1998), pp. 1947–1980.
- [172] Yzelman, Rob H. Bisseling, Dirk Roose, and Karl Meerbergen. “MulticoreBSP for C: A High-Performance Library for Shared-Memory Parallel Programming”. *International Journal of Parallel Programming* 42 (2013), pp. 619–642.
- [173] Olaf Bonorden, Ben Juurlink, Ingo von Otte, and Ingo Rieping. “The Paderborn University BSP (PUB) library”. *Parallel Computing* 29.2 (2003), pp. 187–207.
- [174] Victor Allombert, Frédéric Gava, and Julien Tesson. “Multi-ML: Programming Multi-BSP Algorithms in ML”. *International Journal of Parallel Programming* 45.2 (2017), pp. 340–361.

- [175] Larry McVoy and Carl Staelin. “Lmbench: Portable Tools for Performance Analysis”. In: *USENIX Annual Technical Conference*. ATEC ’96. 1996, pp. 279–294. URL: <http://www.bitmover.com/lmbench/>.
- [176] Samy Al Bahra. “Nonblocking Algorithms and Scalable Multicore Programming”. *ACM Queue* 11.5 (2013), 40:40–40:64. URL: <https://queue.acm.org/detail.cfm?id=2492433>.
- [177] François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst. “hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications”. In: *PDP 2010 - The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing*. 2010.

## Algorithmes d’approximation en optimisation combinatoire

### Livres, thèses et rapports techniques

- [178] Ngoc Le. “Algorithms for the Maximum Independent Set Problem”. PhD thesis. Technische Universität Bergakademie Freiberg, 2014.
- [179] Emile Aarts and Jan Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. John Wiley & Sons, Inc., 1989. ISBN: 0-471-92146-7.
- [180] Dana Ballard, P. Gardner, and M. Srinivas. *Graph Problems and Connectionist Architectures*. Tech. rep. CSD/TR167. University of Rochester, New-York, 1987.

### Articles et actes de conférences

- [181] Zvi Galil. “Efficient Algorithms for Finding Maximum Matching in Graphs”. *ACM Computing Survey* 18.1 (1986), pp. 23–38.
- [182] Michael Luby. “A Simple Parallel Algorithm for The Maximal Independent Set Problem”. In: *17<sup>th</sup> ACM Symposium on Theory Computing*. ACM, 1985, pp. 1–10.
- [183] Alex Pothen and Chin-Ju Fan. “Computing the Block Triangular Form of a Sparse Matrix”. *ACM Transactions on Mathematical Software* 16.4 (1990), pp. 303–324.
- [184] Magnus Halldórsson and Jaikumar Radhakrishnan. “Greed is good: Approximating independent sets in sparse and bounded-degree graphs”. *Algorithmica* 18.1 (1997), pp. 145–163.
- [185] Thomas Feo and Mauricio Resende. “Greedy Randomized Adaptive Search Procedures”. *Journal of Global Optimization* 6.2 (1995), pp. 109–133.
- [186] Mhand Hifi. “A genetic algorithm-based heuristic for solving the weighted maximum independent set and some equivalent problems”. *Journal of the Operational Research Society* 48.6 (1997), pp. 612–622.
- [187] Carlo Mannino and Egidio Stefanutti. “An Augmentation Algorithm for the Maximum Weighted Stable Set Problem”. *Computational Optimization and Applications* 14.3 (1999), pp. 367–381.
- [188] Andreas Brandstädt and Vadim Vozin. “A note on  $\alpha$ -redundant vertices in graphs”. *Discrete Applied Mathematics* 108.3 (2001), pp. 301–308.
- [189] Jianer Chen, Iyad Kanj, and Weijia Jia. “Vertex Cover: Further Observations and Further Improvements”. *Journal of Algorithms* 41.2 (2001), pp. 280–301.
- [190] Robert Tarjan. “Decomposition by clique separators”. *Discrete Mathematics* 55.2 (1985), pp. 221–232.
- [191] Raffaele Mosca. “Independent sets in  $(P_6,\text{diamond})$ -free graphs”. *Discrete Mathematics and Theoretical Computer Science* 11 (2009), pp. 125–140.
- [192] Dominic Welsh and Martin Powell. “An upper bound for the chromatic number of a graph and its application to timetabling problems”. *The Computer Journal* 10.1 (1967), pp. 85–86.
- [193] Daniel Brélaz. “New Methods to Color the Vertices of a Graph”. *ACM Communications* 22.4 (1979), pp. 251–256.

- [194] Thomas Coleman and Jorge Moré. “Estimation of sparse hessian matrices and graph coloring problems”. *Mathematical Programming* 28.3 (1984), pp. 243–270.
- [195] Zvi Galil. “Efficient algorithms for finding maximal matching in graphs”. In: *CAAP: Colloquium on Trees in Algebra and Programming*. Springer Berlin Heidelberg, 1983, pp. 90–113.
- [196] Richard Karp and Michael Sipser. “Maximum Matching in Sparse Random Graphs”. In: *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*. 1981, pp. 364–375.
- [197] Jack Edmonds. “Paths, Trees and Flowers”. *Canadian Journal of Mathematics* (1965), pp. 449–467.
- [198] Claude Berge. “Two Theorems in Graph Theory”. *Proceedings of the National Academy of Sciences of the USA* 43.9 (1957), pp. 842–844.

## Algorithmes de graphes sur architectures manycore

### Livres, thèses et rapports techniques

- [199] Georgios Rokos. “Scalable Multithreaded Algorithms for Mutable Irregular Data with Application to Anisotropic Mesh Adaptivity”. PhD thesis. Imperial College London, 2014.
- [200] Assefaw Gebremedhin. “Parallel Graph Coloring”. MA thesis. University of Bergen, 1999.
- [201] Assefaw Gebremedhin. “Practical Parallel Algorithms for Graph Coloring Problems in Numerical Optimization”. PhD thesis. University of Bergen, Norway, 2003.
- [202] Amy Shoemaker and Sagar Vare. *Edmonds’ blossom algorithm*. Tech. rep. CME 323. Stanford University, 2016.
- [203] Eric Chang and Rutwik Parikh. *A parallel implementation of Edmonds’s Blossom Algorithm using OpenMP*. 2013. URL: <https://github.com/Xelaadryth/Edmonds-Parallel>.

### Articles et actes de conférences

- [204] Umit Çatalyurek, John Feo, Assefaw Gebremedhin, Mahantesh Halappanavar, and Alex Pothen. “Graph Colouring Algorithms for Multi-core and Massively Multithreaded Architectures”. In: *Journal of Parallel Computing*. Vol. 38. Elsevier, 2012, pp. 576–594.
- [205] Assefaw Gebremedhin and Fredrik Manne. “Scalable parallel graph coloring algorithms”. *Concurrency: Practice and Experience* 12.12 (2000), pp. 1131–1146.
- [206] Georgios Rokos, Gerard Gorman, and Paul Kelly. “A Fast and Scalable Graph Coloring Algorithm for Multicore and Manycore Architectures”. In: *Euro-Par 2015: Parallel Processing*. Ed. by Sascha Hunold Jesper Larsson Träff and Francesco Versaci. Springer Berlin Heidelberg, 2015, pp. 414–425.
- [207] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. “R-MAT: A recursive model for graph mining”. In: *In SDM*. 2004.
- [208] Aydin Buluç, John R. Gilbert, and Viral B. Shah. “Implementing Sparse Matrices for Graph Algorithms”. *Graph Algorithms in the Language of Linear Algebra* (2011). Ed. by Jeremy Kepner and John Gilbert.
- [209] Douglas Gregor and Andrew Lumsdaine. “The parallel bgl: A generic library for distributed graph computations”. In: *In Parallel Object-Oriented Scientific Computing (POOSC*. 2005.
- [210] Lori Freitag, Mark Jones, and Paul Plassmann. “The Scalability Of Mesh Improvement Algorithms”. In: *The IMA Volumes in Mathematics and its Applications*. Springer-Verlag, 1998, pp. 185–212.
- [211] Mehmet Deveci, Erik Boman, Karen Devine, and Siva Rajamanickam. “Parallel Graph Coloring for Manycore Architectures”. In: *IEEE International Parallel and Distributed Processing Symposium*. 2016, pp. 892–901.

- [212] Mahantesh Halappanavar, Alex Pothen, Ariful Azad, Fredrik Manne, Johannes Langguth, and Arif Khan. “Codesign Lessons Learned from Implementing Graph Matching on Multithreaded Architectures”. *Computer Journal* 48.8 (2015), pp. 46–55.
- [213] Mahantesh Halappanavar, John Feo, Oreste Villa, Antonino Tumeo, and Alex Pothen. “Approximate weighted matching on emerging manycore and multithreaded architectures”. *Journal of High Performance Computing Applications* 26.4 (2012), pp. 413–430.
- [214] Fredrik Manne and Mahantesh Halappanavar. “New Effective Multithreaded Matching Algorithms”. In: *28th International Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2014, pp. 519–528.
- [215] Ariful Azad, Aydin Buluc, and Alex Pothen. “Computing Maximum Cardinality Matchings in Parallel on Bipartite Graphs via Tree-Grafting”. *IEEE Transactions on Parallel & Distributed Systems* 28.1 (2017), pp. 44–59.
- [216] Ariful Azad, Mahantesh Halappanavar, Sivasankaran Rajamanickam, Erik Boman, Arif Khan, and Alex Pothen. “Multithreaded Algorithms for Maximum Matching in Bipartite Graphs”. In: *IEEE 26th International Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2012, pp. 860–872.
- [217] Johannes Langguth, Ariful Azad, Mahantesh Halappanavar, and Fredrik Manne. “On Parallel Push-relabel Based Algorithms for Bipartite Maximum Matching”. *Parallel Computing* 40.7 (2014), pp. 289–308.
- [218] Mostofa Ali Patwary, Rob Bisseling, and Fredrik Manne. “Parallel Greedy Graph Matching Using an Edge Partitioning Approach”. In: *4th International Workshop on High-level Parallel Programming and Applications*. HLPP ’10. ACM, 2010, pp. 45–54.
- [219] Andrew Lumsdaine, Douglas Gregor, Bruce Hendrickson, and Jonathan Berry. “Challenges in Parallel Graph Processing”. In: *Parallel Processing Letters*. Vol. 17. 2007, pp. 5–20.
- [220] Jarek Nieplocha, Andrès Márquez, John Feo, Daniel Chavarría-Miranda, George Chin, Chad Scherrer, and Nathaniel Beagley. “Evaluating the Potential of Multithreaded Platforms for Irregular Scientific Computations”. In: *Proceedings of the 4th International Conference on Computing Frontiers*. ACM, 2007, pp. 47–58.
- [221] John Feo, David Harper, Simon Kahan, and Petr Konecny. “ELDORADO”. In: *Proceedings of the 2Nd Conference on Computing Frontiers*. ACM, 2005, pp. 28–34.
- [222] Farzad Khorasani, Rajiv Gupta, and Laxmi N. Bhuyan. “Scalable SIMD-Efficient Graph Processing on GPUs”. In: *International Conference on Parallel Architecture and Compilation*. PACT ’15. IEEE Computer Society, 2015, pp. 39–50.
- [223] Bas Fagginger Auer and Rob Bisseling. “A GPU Algorithm for Greedy Graph Matching”. In: *Facing the Multicore-Challenge II*. Ed. by Rainer Keller, David Kramer, and Jan-Philipp Weiss. Springer-Verlag, 2012, pp. 108–119. ISBN: 978-3-642-30396-8.

## PROCES VERBAL DE SOUTENANCE DE DOCTORAT

à déposer dans les 3 jours ouvrés après la soutenance au service de scolarité de l'établissement de préparation du doctorat daté et signé

Monsieur Hoby RAKOTOARIVELO

**ECOLE DOCTORALE :** Sciences et Technologies de l'Information et de la Communication  
**ETABLISSEMENT DE PREPARATION DU DOCTORAT:** université d'Evry-Val-d'Essonne

**Titre de la thèse :** Contributions au co-design de noyaux irréguliers sur accélérateurs manycore : cas du remaillage anisotrope multi-échelle en mécanique des fluides numérique.

**Spécialité de doctorat :** Informatique

Date de soutenance : 6 juillet 2018 Heure : 10h00 Lieu : Petit Amphi Bâtiment IBGBI, 23 boulevard de France, 91000 Évry

### Décision du Jury

- Admission  
 Ajournement

### Avis du Jury sur la reproduction de la thèse

- Thèse dont le dépôt légal peut être finalisé après des modifications mineures\*  
 Corrections majeures demandées par le jury

Si des corrections sont demandées, **membre du Jury désigné par le président** pour vérifier les corrections : Nom, prénom, titre et fonction dans le jury :

Dans ce cas, le président du jury, complète, date et signe le formulaire de vérification des corrections majeures apportées à la thèse et le remet au membre du jury chargé de la vérification.

\* Des modifications mineures seront toujours nécessaires au moins pour préciser le nom du président du Jury sur la page de couverture en vue du 2eme dépôt légal de la thèse.

Civilité, Nom et Prénom	Titre	Fonction dans le Jury	Signature
M. Franck POMMEREAU	Professeur des Universités	Directeur de thèse	selon l'arrêté du 25 Mai 2016, le directeur de thèse participe au Jury mais ne prend pas part à la décision
M. Jean-François REMACLE	Professeur des Universités	Rapporteur <input type="checkbox"/> Président	
M. Frédéric GAVA	Maître de Conférences	Rapporteur	
M. Franck LEDOUX	Directeur de Recherche	Examinateur <input type="checkbox"/> Président	
Mme Hanna KLAUDEL	Professeur des Universités	Examinateur <input checked="" type="checkbox"/> Présidente	
Mme Laure GONNORD	Maître de Conférences	Examinateur	
M. Adrien LOSEILLE	Chargé de Recherche	Examinateur	

Évry, le 6/7/2018

## RAPPORT DE SOUTENANCE DE DOCTORAT

à déposer si possible trois jours ouvrés après la soutenance et au plus tard un mois après la soutenance au service de scolarité de l'établissement de préparation de la thèse, daté et signé

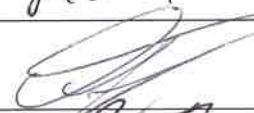
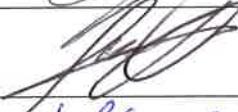
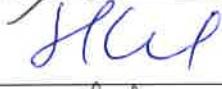
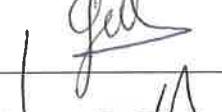
Monsieur Hoby RAKOTOARIVELO

ECOLE DOCTORALE : Sciences et Technologies de l'Information et de la Communication

ETABLISSEMENT DE PREPARATION DU DOCTORAT: université d'Evry-Val-d'Essonne

Le candidat a présenté de façon pédagogique, structurée et claire ses travaux de recherche couvrant deux domaines, d'une part le calcul scientifique (adaptation des mariage), et d'autre part le calcul parallèle haute performance sur architectures multi-coeurs.  
 Il a répondu avec aisance et modestie à de nombreuses questions du jury qui portaient sur l'ensemble des domaines traités. Son exposé et les questions ont permis de mettre en valeur l'originalité de ses solutions ainsi que sa grande maîtrise de l'état de l'art. Le jury a particulièrement apprécié la qualité et l'ampleur du travail réalisé incluant de nombreuses implantations et expérimentations sur des exemples réalistes.

Evry, le 6/7/2018

Civilité, Nom et Prénom	Titre	Fonction dans le Jury	Signature
M. Franck POMMEREAU	Professeur des Universités	Directeur de thèse	
M. Jean-François REMACLE	Professeur des Universités	Rapporteur <input type="checkbox"/> Président	P.O. 
M. Frédéric GAVA	Maître de Conférences	Rapporteur	
M. Franck LEDOUX	Directeur de Recherche	Examinateur <input type="checkbox"/> Président	
Mme Hanna KLAUDEL	Professeur des Universités	Examinateur <input checked="" type="checkbox"/> Présidente	H.K. 
Mme Laure GONNORD	Maître de Conférences	Examinateur	L.G. 
M. Adrien LOEILLE	Chargé de Recherche	Examinateur	A.L. 

CONTRIBUTIONS AU CO-DESIGN DE NOYAUX IRRÉGULIERS SUR ARCHITECTURES MANYCORE.  
CAS DU REMAILLAGE ANISOTROPE MULTI-ÉCHELLE EN MÉCANIQUE DES FLUIDES NUMÉRIQUE.

**Résumé :** La simulation numérique d'écoulements complexes telles que les turbulences ou la propagation d'ondes de choc implique un temps de calcul conséquent pour une précision industrielle acceptable. Pour accélérer ces simulations, deux recours peuvent être combinés : l'adaptation de maillages afin de réduire le nombre de points d'une part, et le parallélisme pour absorber la charge de calcul d'autre part. Néanmoins réaliser un portage efficient des noyaux adaptatifs sur des architectures massivement parallèles n'est pas triviale. Non seulement chaque tâche relative à un voisinage local du domaine doit être propagée, mais le fait de traiter une tâche peut générer d'autres tâches potentiellement conflictuelles. De plus, les tâches en question sont caractérisées par une faible intensité arithmétique ainsi qu'une faible réutilisation de données déjà accédées. Par ailleurs, l'avènement de nouveaux types de processeurs dans le paysage du calcul haute performance implique un certain nombre de contraintes algorithmiques. Dans un contexte de réduction de la consommation électrique, ils sont caractérisés par de multiples cores faiblement cadencés et une hiérarchie mémoire profonde impliquant un coût élevé et asymétrique des accès-mémoire. Ainsi maintenir un rendement optimal des cores implique d'exposer un parallélisme très fin et élevé d'une part, ainsi qu'un fort taux de réutilisation de données en cache d'autre part. Ainsi la vraie question est de savoir comment structurer ces noyaux data-driven et data-intensive de manière à respecter ces contraintes ?

Dans ce travail, nous proposons une approche qui concilie les contraintes de localité et de convergence en termes d'erreur et qualité de mailles. Plus qu'une parallelisation, elle s'appuie une re-conception des noyaux guidée par les contraintes hardware en préservant leur précision. Plus précisément, nous proposons des noyaux locality-aware pour l'adaptation anisotrope de variétés différentielles triangulées, ainsi qu'une parallelisation lock-free et massivement multithread de noyaux irréguliers. Bien que complémentaires, ces deux axes proviennent de thèmes de recherche distinctes mêlant informatique et mathématiques appliquées. Ici, nous visons à montrer que nos stratégies proposées sont au niveau de l'état de l'art pour ces deux axes.

**Mots-clés :** noyaux irréguliers, adaptation anisotrope de maillages, calcul parallèle, machines manycore.

A CODESIGN APPROACH OF IRREGULAR KERNELS ON MANYCORE ARCHITECTURES.  
CASE OF MULTI-SCALE ANISOTROPIC REMESHING IN COMPUTATIONAL FLUID DYNAMICS.

**Abstract :** Numerical simulations of complex flows such as turbulence or shockwave propagation often require a huge computational time to achieve an industrial accuracy level. To speedup these simulations, two alternatives may be combined : mesh adaptation to reduce the number of required points on one hand, and parallel processing to absorb the computation workload on the other hand. However efficiently porting adaptive kernels on massively parallel architectures is far from being trivial. Indeed each task related to a local vicinity need to be propagated, and it may induce new conflictual tasks though. Furthermore, these tasks are characterized by a low arithmetic intensity and a low reuse rate of already cached data. Besides, new kind of accelerators have arised in high performance computing landscape, involving a number of algorithmic constraints. In a context of electrical power consumption reduction, they are characterized by numerous underclocked cores and a deep hierarchy memory involving asymmetric expensive memory accesses. Therefore, kernels must expose a high degree of concurrency and high cached-data reuse rate to maintain an optimal core efficiency. The real issue is how to structure these data-driven and data-intensive kernels to match these constraints ?

In this work, we provide an approach which conciliates both locality constraints and convergence in terms of mesh error and quality. More than a parallelization, it relies on redesign of kernels guided by hardware constraints while preserving accuracy. In fact, we devise a set of locality-aware kernels for anisotropic adaptation of triangulated differential manifold, as well as a lock-free and massively multithread parallelization of irregular kernels. Although being complementary, those axes come from distinct research themes mixing informatics and applied mathematics. Here, we aim to show that our devised schemes are as efficient as the state-of-the-art for both axes.

**Keywords :** irregular kernels, anisotropic mesh adaptation, parallel processing, manycore machines.

