PolyPublie
Polytechnique Montréal

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Titre: Title:	Technique adaptative pour la méthode de Galerkin sans maillage en mécanique des milieux continus
Auteur: Author:	Paul Michelon
Date:	2020
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Michelon, P. (2020). Technique adaptative pour la méthode de Galerkin sans maillage en mécanique des milieux continus [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/5391/

Document en libre accès dans PolyPublie
Open Access document in PolyPublie

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/5391/
Directeurs de recherche: Advisors:	Benoît Ozell
Programme: Program:	Génie informatique

Ce fichier a été téléchargé à partir de PolyPublie, le dépôt institutionnel de Polytechnique Montréal
This file has been downloaded from PolyPublie, the institutional repository of Polytechnique Montréal

https://publications.polymtl.ca

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Technique adaptative pour la méthode de Galerkin sans maillage en mécanique des milieux continus

PAUL MICHELON
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Août 2020

© Paul Michelon, 2020.

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Technique adaptative pour la méthode de Galerkin sans maillage en mécanique des milieux continus

présenté par **Paul MICHELON**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Farida CHERIET, présidente
Benoit OZELL, membre et directeur de recherche
Steven DUFOUR, membre

DÉDICACE

À mes parents sans qui cette expérience au Québec n'aurait pas été possible.

REMERCIEMENTS

Je remercie mon directeur de recherche Benoît Ozell pour m'avoir accepté dans son laboratoire, ainsi qu'OSSimTech et le gouvernement canadien pour avoir financé ma maîtrise.

Je tiens en particulier à remercier Vincent Magnoux pour son aide précieuse tout au long de ma maîtrise, et surtout pendant les mois de confinement !

RÉSUMÉ

La Méthode de Galerkin Sans Maillage (MGSM) est une variante de la Méthode des Éléments Finis (MEF) utilisant un nuage de points non connectés par un maillage. Nous utilisons cette technique de l'analyse numérique pour approximer des problèmes d'élasticité linéaire se posant lors de la réalisation d'un simulateur de chirurgie. Ce travail traite d'un module d'adaptation permettant de relocaliser les degrés de liberté pour augmenter l'efficience de la MGSM. Il est composé d'un estimateur d'erreur reposant sur la décomposition polaire du gradient du déplacement. Un second sous-module consiste en un critère d'adaptation permettant de décider quelles parties de l'objet subdiviser ou décimer selon l'estimation de l'erreur. Enfin, on transforme ces parties en ajoutant ou retirant des degrés de liberté en respectant la structure régulière adoptée par le simulateur. Nous comparons ensuite le modèle avec et sans adaptabilité à une solution analytique dans une configuration de poutre encastrée soumise à la gravité. Nous effectuons en outre un test vérifiant que l'adaptation réagit convenablement à des déformations dynamiques. Nos résultats montrent que notre méthode n'est pas gourmande en temps de calcul, mais qu'elle augmente l'erreur contrairement à notre hypothèse de départ. Cela est dû aux discontinuités entre les milieux ayant été adaptés à différents niveaux. Autrement, tous nos objectifs relatifs au fonctionnement de l'adaptation, c'est-à-dire l'estimation de l'erreur, le critère d'adaptation, la subdivision et la décimation sont fonctionnels et validés.

ABSTRACT

The Element Free Galerkin Method (EFG) is a variant of the Finite Element Method (FEM) using a cloud of points instead of a mesh. We use this numerical analysis technique to solve linear elasticity problems arising during the realization of a surgery simulator. This work deals with an adaptation module allowing to relocate the degrees of freedom to increase the efficiency of EFG. It is composed of an error estimator based on the polar decomposition of the displacement gradient. A second sub-module consists of an adaptation criterion making it possible to decide which parts of the object to refine or decimate according to the estimate of the error. Finally, these parts are transformed by adding or removing degrees of freedom while respecting the regular structure adopted by the simulator. We then compare to an analytical solution the model with and without adaptivity in a fixed beam configuration subjected to gravity. We also carry out a test verifying that the adaptation reacts suitably to dynamic deformations. Our results show that that our method is not greedy in computing time but that it increases the error, contrary to our initial hypothesis. This is due to the discontinuities in the parts of the object having been adapted at different levels. Otherwise, all our objectives relating to the functioning of the adaptation, which are the error estimator, the adaptation criterion, the subdivision and the decimation are functional and validated.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

DDL	Degrés De Liberté
PI	Points d'Intégration
EFG	Element Free Galerkin
MLS	moindres carrés mobiles, Moving Least Squares
MLPG	méthode Meshless-Local-Petrov-Galerkin
MEF	méthode des éléments finis
SOFA	Simulation Open Framework Architecture

LISTE DES SYMBOLES

$\boldsymbol{x}$	Point de l'espace $\mathbb{R}^3$
t	Variable temporelle ($t \geq 0$)
Ω	Domaine occupé par l'objet non déformé
τ	Frontière de Ω
∇	Opérateur Nabla
∇_x	Gradient spatial
σ	Tenseur des contraintes
u	Vecteur de déformation
u^h	Approximation de u
ϵ	Tenseur des déformations
ρ	Densité de masse volumique
f^{ext}	Forces externes
ω	Fonctions de test
λ, μ	Coefficients de Lamé
ϕ	Fonctions de forme
Φ	Fonctions de base
M	Matrice de masse
K	Matrice de rigidité
$\|.\|$	Cardinal ou valeur absolue ou semi-norme
Δ	Laplacien

LISTE DES ANNEXES

CHAPITRE 1 INTRODUCTION

L'analyse numérique est un domaine des mathématiques proche de l'informatique dont l'objet est d'approximer numériquement des problèmes de l'analyse mathématique. Parmi les méthodes faisant partie de cette discipline, la plus connue et populaire est probablement la Méthode des Éléments Finis (MEF), décrite pour la première fois en 1956 par Turner et al. [1]. De nombreuses variantes de cette théorie ont vu le jour depuis, notamment la Méthode de Galerkin Sans maillage (MGSM), sur laquelle nous allons porter notre attention dans ce mémoire. Cette dernière a été développée par Belytschko [2] en 1996 et reste encore moins répandue que la MEF. La principale différence entre ces deux méthodes est que la MEF utilise le plus souvent un maillage de tétraèdres, alors que la MGSM est une technique dite sans maillage, dont le fonctionnement repose sur un nuage de Degrés De Liberté (DDL) et une grille d'intégration sous-jacente.

Les méthodes d'analyse numériques sont utilisées dans tous les domaines de l'ingénierie où il est nécessaire d'approximer des équations aux dérivées partielles. Elles permettent d'effectuer des simulations physiquement réalistes à partir des équations du sous-domaine de la physique le plus proche du problème considéré. Dans notre cas, nous travaillerons sur un simulateur d'opérations chirurgicales, ce qui nous demandera d'utiliser le formalisme de la mécanique des milieux continus. C'est sur ce champ de la physique que se base le simulateur en temps réel de Vincent Magnoux [3] auquel nous viendrons apporter des modifications.

La raison pour laquelle la MGSM a été préférée à la MEF est qu'il a été jugé plus simple d'effectuer des découpes de liens dans une méthode sans maillage, car on n'a pas à régénérer de maillage dans ce cas. La découpe d'un objet est en effet un module important du simulateur de Vincent Magnoux [3] mais nous ne traiterons pas de ce sujet ici. Nous nous concentrerons uniquement sur les déformations élastiques.

1.1 Éléments de la problématique

Généralement, lorsqu'on utilise la MEF dans des problèmes en temps réel , on dispose d'un maillage fixe dans le temps sur lequel on vient approximer nos équations. Similairement, les nuages de DDL sont en général fixes pour la MGSM. Nous souhaiterions, dans la MGSM, pouvoir adapter la densité de DDL pour améliorer l'efficience de notre simulation au sens du rapport entre la précision et les ressources en temps de calcul. Cela demanderait de

savoir dans un premier temps où l'erreur est plus importante, puis d'y allouer plus de DDL. Symétriquement, nous souhaiterions pouvoir retirer des DDL dans les zones moins affectées par l'erreur. Au total, il serait préférable de ne pas dépasser un certain nombre de DDL fixé arbitrairement par l'utilisateur.

Cela présuppose qu'une variation de la densité suivant une estimation de l'erreur devrait donner un meilleur résultat de l'erreur globale.

1.2 Objectifs de recherche

L'objectif principal de cette recherche est de développer une méthode adaptative permettant d'améliorer l'efficience de la MGSM. Elle devra être compatible avec le programme déjà développé pour le simulateur chirurgical.

Les objectifs spécifiques sont les suivants :

- O1 : Implémentation d'un estimateur d'erreur ;
- O2 : Implémentation d'un critère de subdivision ;
- O3 : Implémentation d'un critère de décimation ;
- O4 : Subdivision du maillage de DDL et de cellules d'intégration ;
- O5 : Décimation du maillage de DDL et de cellules d'intégration ;
- O6 : Calcul des composantes des DDL et Points d'Intégration (PI) après subdivision ;
- O7 : Calcul des composantes des DDL et PI après décimation.
- O8 : Vérification de la méthodologie développée avec les hypothèses suivantes :
  - Hypothèse H1 : la nouvelle méthode ne nécessite pas pas plus de 10% de temps de calcul supplémentaire par rapport à l'algorithme non adaptatif pour un même nombre de DDL.
  - Hypothèse H2 : la nouvelle méthode permet de réduire l'erreur d'au moins 10% par rapport à la version non adaptative de l'algorithme pour un même nombre de DDL.

1.3 Plan du mémoire

Nous découperons le mémoire en quatre parties :

- une revue de littérature analysant les différentes méthodes adaptatives déjà mises en oeuvre pour s'en inspirer et chercher à les améliorer ou pour les adapter à notre

problématique. C'est ici que nous ferons un récapitulatif théorique du fonctionnement du simulateur ;

— une partie traitant de la méthodologie de notre technique d'adaptation, détaillant le fonctionnement de chaque sous-module ;

— les résultats et leur discussion sous la forme de deux scénarios complémentaires ;

— et enfin la conclusion de ces recherches.

CHAPITRE 2 REVUE DE LITTÉRATURE

2.1 Introduction

Les méthodes adaptatives sont des outils essentiels pour améliorer la précision des approximations numériques dans de nombreux problèmes de la physique. Ces techniques sont parfois implémentées dans le cadre de la méthode des éléments finis pour en améliorer l'efficience. Elles y rencontrent toutefois des difficultés lors de grandes déformations élastiques en produisant des distorsions du maillage [4]. Pour éviter cet inconvénient, il est possible d'avoir recours aux techniques sans maillage. Ces dernières font plutôt appel à un ensemble dispersé de noeuds qui est plus facilement modifiable.

Notre étude se concentrera principalement sur la MGSM telle que définie par Belytschko [5]. Nous présenterons tout d'abord le modèle mathématique implémenté dans le simulateur [3,6] où nous définirons toutes les notations que nous utiliserons dans la suite du mémoire. Cela consistera en la définition de la forme variationnelle, la discrétisation spatiale, la discrétisation temporelle et l'assemblage des matrices. Ensuite, nous nous intéresserons aux estimateurs d'erreurs de la littérature, puis aux stratégies de raffinement, et enfin aux transferts de données entre adaptations successives.

2.2 Modèle mathématique

Nous commençons par rappeler les bases théoriques sur lesquelles s'appuie notre méthode.

2.2.1 Forme quasi variationnelle

Soit $\Omega \subset \mathbb{R}^3$ un domaine représentant un solide à l'état non déformé, $\boldsymbol{x} \in \Omega$ la variable d'espace, $t \in \mathbb{R}_+$ la variable de temps. On notera le gradient spatial ∇_x.

La forme forte du problème provient de la seconde loi de la dynamique :

$$\rho(\boldsymbol{x})\ddot{u}(\boldsymbol{x},t) = f^{ext}(\boldsymbol{x}) + (\nabla_x \cdot \sigma(u))(\boldsymbol{x},t) \tag{2.1}$$

où $\rho : \Omega \to \mathbb{R}$ est la densité de masse volumique de l'objet, $u : \Omega \times \mathbb{R} \to \mathbb{R}^3$ est le champ de déformation, $r : \mathbb{R}^3 \to \mathbb{R}^3$ la densité des forces extérieures (nous ne considérons qu'un

champ stationnaire pour simplifier), et $\sigma : (\Omega \times \mathbb{R} \to \mathbb{R}^3) \to (\Omega \times \mathbb{R} \to M_{3,3}(\mathbb{R}))$ le tenseur des contraintes.

On pose le tenseur des déformations :

$$\epsilon(u) = \frac{\nabla_x u + \nabla_x u^t}{2} \tag{2.2}$$

En réalité, on n'utilisera pas directement le gradient de la déformation, mais on définira une matrice de corotation pour éviter que les rotations induisent des contraintes dans l'objet, ce qui est la conséquence de la relation linéaire que nous utilisons entre le tenseur des déformations et le gradient de la déformation. On fait donc la décomposition polaire suivante du gradient de la déformation :

$$\nabla_x u = RS \tag{2.3}$$

où R est une matrice de rotation. On remplace ensuite $\nabla_x u$ par :

$$S = R^T \nabla_x u \tag{2.4}$$

Ce détail est d'une certaine importance puisque nous nous baserons sur cette même quantité pour estimer l'erreur dans l'objet. On notera dans la suite du document $\nabla_x u$ pour S.

Nous définissons ensuite l'expression du tenseur des contraintes :

$$\sigma = \lambda tr(\epsilon) I + 2\mu\epsilon \tag{2.5}$$

où λ et μ sont les coefficients de Lamé.

Maintenant que tous les membres de la forme forte sont définis, on cherche la formulation variationnelle. Pour cela, on multiplie l'équation (2.1) par une fonction test appartenant à l'espace de Sobolev $H^1(\Omega)^3$ et on intègre sur Ω :

$$\int_\Omega \rho \ddot{u} \omega \, dv = \int_\Omega (f^{ext} + \nabla_x \cdot \sigma)\omega \, dv \tag{2.6}$$

En intégrant par parties, on peut montrer [7] que la forme variationnelle est la suivante :

$$\int_\Omega \rho\ddot{u} \cdot \omega \, dv = \int_\Omega f^{ext} \cdot \omega \, dv + \int_\tau n \cdot \omega \, ds - \int_\Omega (\lambda(\nabla_x \cdot u)(\nabla_x \cdot \omega) + 2\mu\epsilon(u) : \epsilon(\omega)) \, dv \quad (2.7)$$

où n est le vecteur normal à la surface frontière τ. En réalité, on appelle plutôt cette équation la forme quasi variationnelle, car elle ne prend pas en compte le temps dans les variables à intégrer.

2.2.2 Discrétisation spatiale

Nous utilisons comme méthode de discrétisation la MGSM tirée de l'article original de Belytschko [2] pour transformer notre problème continu dans l'espace en un problème approximable discrètement par un ordinateur. Nous divisons l'objet en m cubes sur lesquels nous diviserons l'intégrale sur Ω. Notons-les Ω_k de telle sorte que $\cup_{k=1}^{m} \Omega_k \approx \Omega$ et que la mesure de l'intersection de deux Ω_k soit nulle. Dans chaque Ω_k, on pose V_k un ensemble de points. On peut les interpréter comme une analogie des noeuds de calcul de la méthode des éléments finis.

Soit $\boldsymbol{x}_k \in \Omega_k$. Nous exprimons l'approximation de u notée u^h sous la forme suivante :

$$u^h(\boldsymbol{x}_k, t) = \sum_{i \in V_k} u_i(t)\phi_{ki}(\boldsymbol{x}_k) \quad (2.8)$$

où $u_i \in \mathbb{R}^3$ est le DDL associé au noeud de calcul i et $\phi_{ki} : \mathbb{R}^3 \to \mathbb{R}$ est la fonction de forme associée au noeud de calcul i dans Ω_k.

Fonctions de forme

Le but est ici de décrire la méthode de construction des fonctions de forme de la MGSM. Celles-ci sont construites selon la méthode des moindres carrés mobiles (Moving Least Squares, MLS).

Pour plus de lisibilité, nous omettrons la variable temporelle dans cette partie.

On pose :

$$u^h(\boldsymbol{x}_k) = \sum_{i \in V_k} u_i\phi_{ki}(\boldsymbol{x}_k) = p^T(\boldsymbol{x}_k)a(\boldsymbol{x}_k), \quad (2.9)$$

où p est un vecteur de monômes et a contient leurs coefficients (variables). Comme ces coefficients sont seulement connus en $\boldsymbol{x}_k$, ils sont notés a_k. Similairement, comme la déformation u est déterminée uniquement en x_i, elle est notée u_i.

La base que nous avons choisie est :

$$p^T(\boldsymbol{x}) = \begin{bmatrix} 1 & x & y & z \end{bmatrix} \tag{2.10}$$

Pour obtenir les coefficients a_k, on minimise la quantité J suivante pour un $\boldsymbol{x}_k \in \Omega_k$ donné :

$$J(\boldsymbol{x}_k) = \sum_{i \in V_k} w_{h(k,\boldsymbol{x}_k)}(\boldsymbol{x}_k - x_i) \left[p^T(\boldsymbol{x}_k) a_k - u_i \right] \tag{2.11}$$

où les x_i sont les positions associées aux DDL u_i et $w_h : \mathbb{R}^3 \to \mathbb{R}$ est une fonction de poids. Nous avons choisi la fonction de poids suivante :

$$w_h \colon \mathbb{R}^+ \to \mathbb{R}^+ \tag{2.12}$$

$$d \mapsto \frac{105}{16\pi h^3} \cdot (1 - 6(\frac{d}{h})^2 + 8(\frac{d}{h})^3 - 3(\frac{d}{h})^4) \tag{2.13}$$

Et :

$$h(k, \boldsymbol{x}_k) = \frac{\sum_{i \in V_k} \|\boldsymbol{x}_k - x_i\|}{card(V_k)} \cdot e \tag{2.14}$$

où e est le facteur d'expansion, choisi arbitrairement.

On obtient un système linéaire :

$$A_k a_k = Bu \tag{2.15}$$

où u est ici le vecteur ordonné des u_i, et :

$$A = \sum_{i \in V_k} w_{h(k,\boldsymbol{x}_k)}(\boldsymbol{x}_k - x_i) p(\boldsymbol{x}_k) p^T(\boldsymbol{x}_k) \tag{2.16}$$

Après calculs, on peut montrer que :

$$\phi_{ki}(\boldsymbol{x}_k) = p^T(\boldsymbol{x}_k) A_k^{-1} w_{h(k,\boldsymbol{x}_k)}(\boldsymbol{x}_k - x_i) p(\boldsymbol{x}_k) \tag{2.17}$$

De manière similaire, on peut trouver le gradient des fonctions de forme :

$$\begin{aligned}
\phi_{ki,x}(\boldsymbol{x}_k) &= \left[p_{,x}^T(\boldsymbol{x}_k) A^{-1} w_{k,x} + p^T(\boldsymbol{x}_k) \left(A_{,x}^{-1} w_k + A^{-1} w_{k,x}(\boldsymbol{x}_k)\right)\right] p(\boldsymbol{x}_k) \\
\phi_{ki,y}(\boldsymbol{x}_k) &= \left[p_{,y}^T(\boldsymbol{x}_k) A^{-1} w_{k,y} + p^T(\boldsymbol{x}_k) \left(A_{,y}^{-1} w_k + A^{-1} w_{k,y}(\boldsymbol{x}_k)\right)\right] p(\boldsymbol{x}_k) \\
\phi_{ki,z}(\boldsymbol{x}_k) &= \left[p_{,z}^T(\boldsymbol{x}_k) A^{-1} w_{k,z} + p^T(\boldsymbol{x}_k) \left(A_{,z}^{-1} w_k + A^{-1} w_{k,z}(\boldsymbol{x}_k)\right)\right] p(\boldsymbol{x}_k)
\end{aligned} \tag{2.18}$$

où $w_k = w_{h(k,\boldsymbol{x}_k)}(\boldsymbol{x}_k - x_i)$. Ce dernier résultat nous est utile car $\nabla u = \sum_{i \in V_k} u_i \cdot \nabla \phi_{ki}$.

Fonctions de base

Finalement, si l'on impose que les fonctions de forme soient restreintes à leur élément et nulles partout ailleurs, il est possible d'exprimer les fonctions de base selon la formule suivante :

$$\Phi_i = \sum_{k=1}^{m} \phi_{ki} \tag{2.19}$$

Si de plus nous notons n le nombre de DDL du système, on obtient cette seconde formulation de u^h :

$$u^h(\boldsymbol{x}, t) = \sum_{i=1}^{n} u_i(t) \Phi_i(\boldsymbol{x}) \tag{2.20}$$

Cette expression simplifiera nos formulations par la suite.

Système discret

En reprenant la forme variationnelle de l'équation (2.7) et en remplaçant u par u^h comme décrit dans l'équation (2.20), et ω par Φ_i, on obtient :

$$\sum_{j=1}^{n} \ddot{u}_i \int_\Omega \rho\Phi_j \cdot \Phi_i \, dv = \int_\Omega f^{ext} \cdot \Phi_i \, dv + \int_\tau t \cdot \Phi_i \, ds - \sum_{j=1}^{n} u_j \int_\Omega \lambda(\nabla \cdot \Phi_j)(\nabla \cdot \Phi_i) + 2\mu\epsilon(\Phi_j) : \epsilon(\Phi_i) \, dv \tag{2.21}$$

Cela nous donne un système de n équations à n inconnues.

En posant la matrice de masse :

$$M_{ij} = \int_\Omega \rho\Phi_j \cdot \Phi_i \, dv \tag{2.22}$$

La matrice de rigidité :

$$K_{ij} = \int_\Omega \lambda(\nabla \cdot \Phi_j)(\nabla \cdot \Phi_i) + 2\mu\epsilon(\Phi_j) : \epsilon(\Phi_i) \, dv \tag{2.23}$$

Les forces externes :

$$f_i^{ext} = \int_\Omega f^{ext} \cdot \phi_i \, dv \tag{2.24}$$

Et les conditions naturelles :

$$s_i = \int_\tau t \cdot \Phi_i \, ds \tag{2.25}$$

On obtient le système :

$$M\ddot{u} = Ku + f^{ext} + s \tag{2.26}$$

où u, f^{ext} et s sont les vecteurs construits à partir des u_i, f_i et s_i.

2.2.3 Discrétisation temporelle

Le système Scyther utilise un schéma d'Euler implicite [1] pour transformer le problème continu en temps en un problème discret approximable par un ordinateur :

1. Un schéma d'ordre 2 aurait pu être utilisé car il n'est pas beaucoup plus complexe et donne de meilleures performances.

$$M\Delta\dot{u} = (Ku(\boldsymbol{x}, t + \Delta t) + s + f^{ext})\Delta t$$

où $\Delta\dot{u} = \dot{u}(t + \Delta t) - \dot{u}(t)$ et $\Delta t \in \mathbb{R}$ est le pas de temps.

En développant on obtient :

$$Ku(\boldsymbol{x}, t + \Delta t) = Ku + K\Delta u \tag{2.27}$$

où $u(\boldsymbol{x}, t) = u$ et $\Delta u = \dot{u}(t + \Delta t) - u(t)$.

on obtient ensuite :

$$M\Delta\dot{u} = (Ku + K\Delta u + s + f^{ext})\Delta t \tag{2.28}$$

En développant $\Delta u = \Delta t(\dot{u} + \Delta\dot{u})$ où $\Delta\dot{u} = \dot{u}(\boldsymbol{x}, t + \Delta t) - \dot{u}(\boldsymbol{x}, t)$ et $\dot{u}(\boldsymbol{x}, t) = \dot{u}$, on arrive à :

$$(M - \Delta t^2 K)\Delta\dot{u} = \Delta t(Ku + s + f^{ext}) + \Delta t^2 K\dot{u} \tag{2.29}$$

Maintenant que nous avons déterminé l'équation décrivant l'évolution du système, nous utilisons la méthode des gradients conjugués pour résoudre ce système linéaire et trouver $\Delta\dot{u}$. Une fois ceci effectué, on peut calculer les nouvelles positions avec $\Delta u = \Delta t(\dot{u} + \Delta\dot{u})$.

2.2.4 Assemblage des matrices

Avant de pouvoir résoudre l'équation (2.29), il convient de construire K, s, f^{ext} et M [2].

Comme nous ne pouvons pas intégrer analytiquement, nous utilisons une quadrature de Gauss d'ordre 1, qui évalue la valeur de l'intégrande au centre du cube $\boldsymbol{x}_k$ et multiplie le résultat par le volume.

On peut ainsi calculer la matrice de rigidité K :

2. Comme nous utilisons les gradients conjugués, on ne calcule pas directement K mais $K\Delta\dot{u}$, de même pour M qui est calculé par l'intermédiaire de $M\Delta\dot{u}$. Les calculs restent cependant similaires.

$$K_{ij} = \sum_{k=1}^{m} (\lambda(\nabla_x \cdot \Phi_j)(\nabla_x \cdot \Phi_i) + 2\mu\epsilon(\Phi_j) : \epsilon(\Phi_i))(\boldsymbol{x}_k) \int_{\Omega_k} dv \tag{2.30}$$

Les autres membres de l'équation (2.29) font l'objet d'approximations :

$$s_i = \int_{\tau} t \cdot \Phi_i \, ds \approx 0 \tag{2.31}$$

Cela correspondrait à une imposition à 0 des fonctions de base sur la frontière, ce qui est une condition naturelle. Dans ce cas, les valeurs des u_i (conditions essentielles) sont libres. On note cependant qu'on force également les valeurs des u_i dans notre méthodologie après leur calcul ce qui se rapproche d'une imposition de conditions essentielles. Il existe des méthodes d'imposition de conditions aux limites pour la MGSM comme les multiplicateurs de Lagrange qui sont théoriquement plus appropriés. Nous en discutons dans l'annexe A.

$$f_i^{ext} = \sum_{k=1}^{m} \int_{\Omega_k} f^{ext} \cdot \Phi_i \approx r_i \tag{2.32}$$

où les r_i sont fixés directement par nous-mêmes. Nous ne considérons pour nos tests que la gravité, ce qui donne pour chaque DDL i :

$$r_i = m(i) \cdot g \tag{2.33}$$

où g est le vecteur d'accélération de pesanteur.

Pour les masses, l'approximation est la suivante :

$$M_{ii} = \sum_{k=1}^{m} \int_{\Omega_k} \rho\Phi_i \cdot \Phi_i \, dv \approx \sum_{k=1}^{m} \Phi_i(\boldsymbol{x}_k) \cdot \rho \cdot \int_{\Omega_k} dv \tag{2.34}$$

Et si $i \neq j$:

$$M_{ij} \approx 0$$

2.3 Principes fondamentaux des méthodes adaptatives

Pour toute méthode adaptative, il existe 3 étapes essentielles :

1. Estimation locale de l'erreur ;
2. Stratégie de raffinement ;
3. Transfert de données entre discrétisations consécutives.

Nous présenterons par la suite les différentes méthodes utilisées dans ces trois étapes. Finalement, nous ferons la synthèse des meilleures approches et nous élaborerons notre hypothèse en construisant notre propre méthode.

2.4 Estimateurs de l'erreur

2.4.1 Estimation par différence du contraintes entre les noeuds et les points d'intégration

Ullah et al. [8] proposent, dans le cas linéaire élastique, de calculer l'estimation de l'erreur de discrétisation des contraintes σ^e comme la différence entre les contraintes aux points d'intégration de Gauss $\sigma^h(\boldsymbol{x})$ et le champ des contraintes projetées à partir des noeuds $\sigma^p(\boldsymbol{x})$:

$$\sigma^e(\boldsymbol{x}) = \sigma^p(\boldsymbol{x}) - \sigma^h(\boldsymbol{x})$$

Dans le cas de la MGSM, les contraintes en tout point se calculent comme suit :

$$\sigma^h(\boldsymbol{x}) = \sum_{d \in V_{k(x)}(x)} DB_d u_d$$

où $k(x)$ est la fonction qui à un x associe l'index k pour lequel $x \in \Omega_k$.

En 2D, on définit D et B_{kd} comme suit :

$$B_i = \begin{bmatrix} \phi_{ki,x} & 0 \\ 0 & \phi_{ki,y} \\ \phi_{ki,y} & \phi_{ki,x} \end{bmatrix} \tag{2.35}$$

$$D = \frac{\bar{E}}{1-\bar{\nu}^2}\begin{bmatrix} 1 & \bar{\nu} & 0 \\ \bar{\nu} & 1 & 0 \\ 0 & 0 & \frac{1-\bar{\nu}}{2} \end{bmatrix} \tag{2.36}$$

où :

$$\bar{E} = \begin{cases} E & \text{pour des contraintes planes} \\ \dfrac{E}{1-\nu^2} & \text{pour des déformations planes} \end{cases} \tag{2.37}$$

$$\bar{\nu} = \begin{cases} \nu & \text{pour des contraintes planes} \\ \dfrac{\nu}{1-\nu^2} & \text{pour des déformations planes} \end{cases} \tag{2.38}$$

$$\phi_{ki,x} = \frac{\partial \phi_{ki}}{\partial x} \tag{2.39}$$

$$\phi_{ki,y} = \frac{\partial \phi_{ki}}{\partial y} \tag{2.40}$$

où ν est le coefficient de Poisson et E est le module d'élasticité.

Les contraintes projetées sont données par :

$$\sigma^p(\boldsymbol{x}) = \sum_{d \in V_{k(x)}(x)} \phi_{kd}(\boldsymbol{x}) \sigma^h(R(d))$$

L'erreur pour une cellule individuelle et pour le domaine entier peut alors être trouvée en utilisant la norme appropriée :

$$\|e\| = \left[\int_\Omega |(\sigma^e(\boldsymbol{x}))^T D^{-1} (\sigma^e(\boldsymbol{x}))| d\Omega \right]^{\frac{1}{2}}$$

On peut utiliser l'erreur relative :

$$\eta = \frac{\|e\|}{\|U\|} \times 100$$

où $\|U\|$ est la norme de l'énergie calculée par :

$$\|U\| = \left[\int_\Omega |(\sigma^p(\boldsymbol{x}))^T D^{-1} (\sigma^p(\boldsymbol{x}))| d\Omega \right]^{\frac{1}{2}}$$

Il n'y a ici pas de garanties théoriques qui nous indiquent si l'erreur réelle est bien liée à cet estimateur, bien qu'il semble intuitif de lier l'estimateur à une mesure des contraintes.

2.4.2 Estimation par les fonctions MLS

Gavete et al. [9] présentent une procédure d'approximation de l'erreur pour la MGSM.

Soit f la fonction à déterminer de notre forme forte. Cela peut être le déplacement, la température ou toute autre grandeur selon le problème. Soit f^h son approximation par la MGSM.

Gavete et al. [9] proposent une estimation de l'erreur a posteriori comme la différence de deux approches : la première utilisée dans la MGSM pour calculer les gradients et la seconde calculée à partir des fonctions de forme MLS utilisant un développement de Taylor :

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_0) + \frac{\partial f}{\partial x}(x - x_0) + \frac{\partial f}{\partial y}(y - y_0) + O((x - x_0)^2 + (y - y_0)^2)$$

On se limitera ici au premier ordre en 2D, mais nous pouvons généraliser.

Considérons la norme :

$$B = \sum_{i=1}^{N} \left[\left[f_0 - f_i + (x_i - x_0)\frac{\partial f}{\partial x} + (y_i - y_0)\frac{\partial f}{\partial y} \right] w_i \right]^2$$

où $f_i = f(\boldsymbol{x}_i)$ et $f_0 = f(\boldsymbol{x}_0)$ sont déterminées avec la MGSM et les w_i sont les poids de cette dernière méthode. L'index i varie de 1 à N, le nombre de DDL que nous associerons au point d'intégration sur lequel nous ferons nos intégrales d'erreur. Les x_i sont les positions à l'état initial des DDL et x_0 est une position à l'état initial des PI.

La solution est obtenue en minimisant la norme B :

$$\frac{\partial B}{\partial Df} = 0$$

où :

$$Df^T = \left[f_0, \frac{\partial f_0}{\partial x}, \frac{\partial f_0}{\partial y} \right] \tag{2.41}$$

où l'équation (2.41) peut correspondre à n'importe quel point d'intégration du domaine.

On arrive à un ensemble de trois équations à trois inconnues pour chaque noeud :

$$\begin{bmatrix} \sum w_i^2 & \sum w_i^2 h_i & \sum w_i^2 k_i \\ \sum w_i^2 h_i & \sum w_i^2 h_i^2 & \sum w_i^2 h_i k_i \\ \sum w_i^2 h_i^2 k_i & \sum w_i^2 h_i k_i & \sum w_i^2 h_i^2 k_i^2 \end{bmatrix} \begin{bmatrix} f_0 \\ \frac{\partial f_0}{\partial x} \\ \frac{\partial f_0}{\partial y} \end{bmatrix} = \begin{bmatrix} \sum f_i w_i^2 \\ \sum f_i w_i^2 h_i \\ \sum f_i w_i^2 k_i \end{bmatrix} \tag{2.42}$$

où $h_i = (x_i - x_0)$ et $k_i = (y_i - y_0)$.

Après avoir calculé les f_i sur les noeuds du domaine avec les approximations de la MGSM, considérons $r > 1$ points dans chaque quadrant dans le voisinage de chacun des points d'intégration (en les prenant pour origine des coordonnées) tel qu'illustré à la figure 2.1 (tirée de [9]). Ces DDL seront les i dans les sommes de l'équation (2.42).

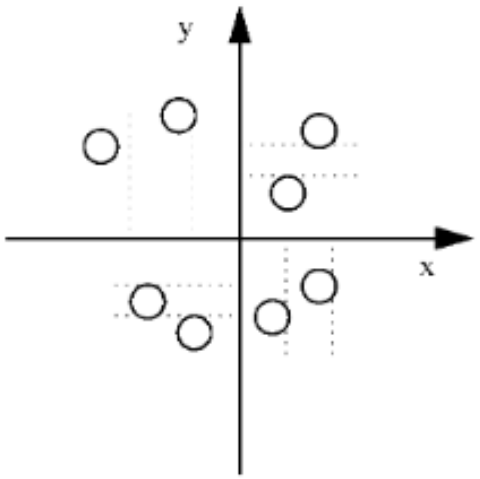

Figure 2.1 Le critère des quatre quadrants pour $r = 2$

L'erreur en norme d'énergie est calculée en prenant en compte le fait que pour chaque point d'intégration nous avons le gradient de deux manières différentes : une par la MGSM (∇f), et l'autre par la formule 2.41 (par l'intermédiaire des dérivées partielles dans la quantité Df). On peut ensuite calculer les contraintes σ et σ^e à partir des gradients. Les formules sur un élément Ω_k pour l'erreur $\|e\|^2_{\Omega_k}$ et pour la quantité $\|w\|^2_{\Omega_k}$ qui permettront par la suite d'avoir une erreur relative sont :

$$\|e\|^2_{\Omega_k} = \left[\int_{\Omega_k} (\nabla f - \nabla f^e)^T (\sigma - \sigma^e) d\Omega \right]$$

$$\|w\|^2_{\Omega_k} = \left[\int_{\Omega_k} (\nabla f^e)^T \sigma^e d\Omega \right]$$

On peut maintenant déterminer la norme sur tout le domaine à partir des sous-domaines :

$$\|e\|^2_{\Omega} = \sum_k \|e\|^2_{\Omega_k}, \quad \|w\|^2_{\Omega} = \sum_k \|w\|^2_{\Omega_k}$$

L'erreur relative et l'indice d'effectivité sont définis comme :

$$\eta = \frac{\|e\|}{\|w\|}$$

$$\theta = \frac{\|e\|}{\|E\|}$$

où $\|E\|$ est l'erreur exacte quand on peut la calculer, c'est-à-dire quand nous avons la solution analytique (ce qui en pratique n'est jamais le cas, mais cela peut servir pour effectuer des vérifications).

Ces quantités peuvent être établies sur tout le domaine ou sur une cellule d'intégration. La déviation de l'indice d'effectivité pour l'estimateur d'erreur sur le domaine complet ou pour chaque cellule d'intégration est définie comme : $d_i = |1 - \theta_i|$

On peut objecter à cette méthode qu'elle ne donne pas réellement un estimateur de l'erreur au sens d'une norme de la différence entre l'approximation MGSM et la solution. De plus,

elle nécessite la résolution de systèmes avant d'avoir les estimations. La recherche des DDL proches d'un PI par la méthode des quadrants nous est en outre inutile puisque notre réseau de noeuds est régulier.

2.4.3 Estimation à partir du tenseur des déformations

Luo et al. [10] proposent quant à eux une procédure d'adaptation basée sur le tenseur des déformations dans la MGSM.

L'énergie potentielle est utilisée pour déterminer quand arrêter l'adaptation du nuage de noeuds. Pour des problèmes élasto-statiques linéaires, l'énergie potentielle totale est donnée par la formule suivante :

$$\Pi = \Pi_s + \Pi_c = \frac{1}{2}\int_\Omega \sigma^T \epsilon \, dV - (\int_\Omega (f^{ext})^T u \, dV + \int_\tau (f^{ext} \cdot n)^T u \, dS)$$

où n est la normale à la surface τ. $\prod_s$ et $\prod_c$ représentent l'énergie de déformation totale et l'énergie potentielle externe.

En discrétisant, on obtient :

$$\widetilde{\Pi} = \widetilde{\Pi}_s + \widetilde{\Pi}_c = \frac{1}{2}\bar{u}^T K \bar{u} - \bar{u}^T F$$

où $\bar{u}$ est le vecteur représentant le déplacement des DDL, K est la matrice de rigidité et F est le vecteur de la force externe équivalente.

En raffinant le modèle, on produit une série d'approximations pour les énergies potentielles $\widetilde{\Pi}_s^i (i = 1 \ldots n)$ correspondant à une série de configurations du nuage de points $M_i (i = 1 \ldots n)$ où n est le nombre d'étapes d'adaptation.

Une condition nécessaire est que l'énergie converge vers la valeur réelle. Autrement dit :

$$\forall \delta > 0, \exists K > 0, \forall k > K, |\widetilde{\Pi}_s^k - \Pi_s| \leq \delta |\Pi_s|$$

$$\forall \delta > 0, \exists K > 0, \forall k > K, |\widetilde{\Pi}_s^{k+1} - \Pi_s| \leq \delta |\Pi_s|$$

Ce qui conduit à :

$$\forall \delta > 0, \exists K > 0, \forall k > K, |\widetilde{\Pi}_s^{k+1} - \widetilde{\Pi}_s^k| \leq \eta |\widetilde{\Pi}_s^{k+1}|$$

où $0 < \eta \leq 2\delta$.

Cette dernière équation donne un critère pour arrêter une boucle adaptative, ce qui est similaire à un estimateur d'erreur global. Localement, il nous faut maintenant déterminer comment raffiner l'objet. Luo et al. [10] proposent d'utiliser l'énergie de déformation bien que d'autres quantités auraient pu servir. Cette énergie de déformation est donnée par la formule suivante :

$$D^{SE} = \frac{1}{2}\sigma^T \epsilon \tag{2.43}$$

Ses variations en 2D sont notées comme suit :

$$G_x^{SED} = \frac{\partial D^{SE}}{\partial x} \tag{2.44}$$

$$G_y^{SED} = \frac{\partial D^{SE}}{\partial y} \tag{2.45}$$

Ces quantités fournissent des informations utiles si le matériau est anisotropique. S'il est isotropique (ce qui est notre cas), on peut en utiliser une moyenne :

$$G^{SED} = \frac{1}{2}(|G_x^{SED}| + |G_y^{SED}|) \tag{2.46}$$

On définit la densité de noeuds D^M comme suit :

$$D^M = \frac{N}{A} \tag{2.47}$$

où N est le nombre de noeuds dans dans la zone délimitée par l'aire (2D) ou le volume (3D) A.

Pour mesurer la richesse du nuage de noeuds par rapport aux variations de l'énergie de déformation, l'intensité du raffinement r_d est définie comme :

$$r_d = \frac{G^{SED}}{D^M} \tag{2.48}$$

Un maillage optimal correspond à une intensité de maillage constante en toute zone. Le maximum et le minimum d'intensité du maillage sont donnés par $R_{\min}$ et $R_{\max}$.

Après avoir obtenu les déplacements aux noeuds, on peut calculer le reste des grandeurs utiles en tout point grâce aux fonctions de forme. On calcule ainsi la variation de l'énergie de déformation. Les extremums d'intensité du maillage sont déterminés, ce qui nous laisse choisir un seuil R_c à partir duquel raffiner ou simplifier le maillage.

$$R_{\min} \leq R_c \leq R_{\max} \tag{2.49}$$

$$\begin{cases} r_d > R_c & \text{raffiner} \\ r_d < R_c & \text{simplifier} \end{cases} \tag{2.50}$$

Pour calculer l'intensité du maillage, l'article [10] propose d'utiliser les cellules d'intégration (Ω_k) comme unité de subdivision de l'espace. Une cellule a une aire A_k et renferme N_k noeuds.

L'énergie de déformation est calculée ainsi :

$$D_i^{SE} = \frac{1}{2}\sigma^T \epsilon \approx \frac{1}{2}\bar{u}^T B_i^T D B_i \bar{u}$$

où D est la matrice de matériau (voir la définition 2.36), et B une matrice de la forme suivante en 2D :

$$B_i^T = \begin{bmatrix} \phi_{i,x} & 0 \\ 0 & \phi_{i,y} \\ \phi_{i,y} & \phi_{i,x} \end{bmatrix} \tag{2.51}$$

La variation de l'énergie de déformation est calculée approximativement comme la moyenne des variations entre les N_k noeuds :

$$G_k^{SED} = \frac{\sum_{i=1}^{N_k} \sum_{j=i+1}^{N_k} |D_i^{SE} - D_j^{SE}|/l_{ij}}{N_k!}$$

où D_i^{SE} est l'énergie de déformation en un noeud i ; l_{ij} est la distance entre deux noeuds i et j. Ainsi, une intensité moyenne sur la cellule k peut être calculée :

$$r_d^k = \frac{G_k^{SED}}{\frac{N_k}{A_k}}$$

On trouve les extremums d'intensité en bouclant sur l'intégralité des cellules d'intégration. À l'aide du seuil de raffinement R_c, on peut calculer le nombre de noeuds par cellule :

$$N_k = \frac{G_k^{SED} A_k}{R_c}$$

Différentes méthodes peuvent être utilisées pour insérer ou retirer des noeuds dans une cellule. Luo et al. [10] ne décrivent pas la méthode, mais renvoient vers Combe et al. [11].

L'idée générale de calculer une intensité de raffinement sous la forme d'un ratio entre la variation de l'énergie de déformation sur la densité du maillage est une bonne façon de comparer les différentes zones de l'objet pour déterminer celles qui nécessitent un raffinement plus important. Néanmoins, un des problèmes de cette méthode est que la quantité R_c est choisie arbitrairement et peut ne pas convenir à tous les problèmes.

2.4.4 Estimation par la borne des fonctions de forme

Combe et al. [11] utilisent l'estimateur d'erreur présenté dans Krongauz et al. [12]. Détaillons son obtention.

L'erreur est donnée par :

$$u^h(\boldsymbol{x}) - u(\boldsymbol{x}) = \sum_{d \in V_{k(x)}(x)} \phi_{kd}(\boldsymbol{x}) u_d - u(\boldsymbol{x})$$

Soit $\boldsymbol{x}_d = (x_d, y_d)$. Supposons que $u(\boldsymbol{x})$ puisse être développé en série de Taylor :

$$\begin{aligned} u(\boldsymbol{x}_d) = u(\boldsymbol{x}) + \frac{\partial u(\boldsymbol{x})}{\partial x}(x_d - x) + \frac{\partial u(\boldsymbol{x})}{\partial y}(y_d - y) + \frac{1}{2}\frac{\partial^2 u(\bar{\boldsymbol{x}})}{\partial x^2}(x_d - x)^2 \\ + \frac{1}{2}\frac{\partial^2 u(\bar{\boldsymbol{x}})}{\partial y^2}(y_I - y)^2 \\ + \frac{\partial^2 u(\bar{\boldsymbol{x}})}{\partial y \partial x}(x_d - x)(y_d - y), \\ \bar{\boldsymbol{x}} = \boldsymbol{x} + \nu(\boldsymbol{x}_d - \boldsymbol{x}), \quad 0 < \nu < 1 \end{aligned} \tag{2.52}$$

On peut montrer que les fonctions de forme sont bornées :

$$|\phi_I(\boldsymbol{x})| \leq c \in \mathbb{R}_+ \tag{2.53}$$

De plus elles ont un support compact :

$$|x_I - x| \leq h_I \tag{2.54}$$

$$|y_I - y| \leq h_I \tag{2.55}$$

Krongauz et al. [12] démontrent la borne suivante sur l'erreur avec la norme L_2 :

$$\begin{aligned} \|u^h - u\|_{L^2(\Omega)} &\leq c h_I^2 \|\frac{1}{2}u_{,xx}(\boldsymbol{x}) + u_{,xy}(\boldsymbol{x}) + \frac{1}{2}u_{,yy}(\boldsymbol{x})\|_{L^2(\Omega)} \\ &\leq c h_I^2 |u(\boldsymbol{x})|_{H^2(\Omega)} \end{aligned} \tag{2.56}$$

On peut également trouver une formule similaire pour borner l'erreur des dérivées. Commençons par considérer l'inégalité suivante :

$|\phi_{I,x}(\boldsymbol{x})| \leq c/h_I$

Cela mène à :

$$\begin{aligned} \|u^h_{,x} - u_{,x}\|_{L^2(\Omega)} &\leq c h_I \|\frac{1}{2}u_{,xx}(\boldsymbol{x}) + u_{,xy}(\boldsymbol{x}) + \frac{1}{2}u_{,yy}(\boldsymbol{x})\|_{L^2(\Omega)} \\ &\leq c h_I |u(\boldsymbol{x})|_{H^2(\Omega)} \end{aligned} \tag{2.57}$$

Ce même résultat vaut aussi pour la dérivée en y.

On a ici notre première garantie théorique sur une borne de l'erreur. On montre 3 points importants :

— l'erreur diminue avec la diminution de la taille des supports des fonctions de forme ;
— elle diminue également avec la borne des fonctions de forme ;
— et enfin elle est d'autant plus petite que la semi-norme H^2 de la solution est petite, c'est-à-dire que ses dérivées partielles secondes sont plus régulières.

Cela indique qu'un raffinement du maillage de noeuds devrait donner de meilleurs résultats de simulation.

2.4.5 Estimateur dans MLPG

Kamranian et al. [4] fournissent un algorithme adaptatif pour une méthode similaire à la MGSM nommée méthode Meshless-Local-Petrov-Galerkin (MLPG) (voir l'annexe B). Ils présentent de nombreuses considérations théoriques avant d'arriver à la définition de l'estimateur d'erreur. Nous formulerons ici uniquement les conclusions.

Soit u la solution exacte et u^h la solution MLPG, $x \subset \Omega$ un ensemble quasi uniforme (défini précisément dans l'article), $\widetilde{V} = Vect(\phi_i(\boldsymbol{x}))_{i=1}^N$ avec les fonctions de forme ϕ_i, et r_i leurs rayons de support, on a la borne de l'erreur a posteriori suivante :

$$\|u - u^h\|_{H^1(\Omega)} \leq C(\sum_{i=1}^{N} r_i^2 \|R\|^2_{L^2(B_i)})^{1/2}$$

Avec le résidu $R = f - \mathcal{L}u^h$ (voir la définition de $\mathcal{L}$ dans l'annexe) et C une constante.

Comme dans Combe et al. [11], on retrouve une relation entre les dérivées secondes (le Laplacien), le support des fonctions de forme et l'erreur. L'erreur est ici globale, mais on peut créer des estimateurs locaux approchés en restreignant la dernière formule.

2.5 Stratégies de Raffinement

2.5.1 Subdivision régulière

Ullah et al. [8] disposent leurs DDL sur une grille régulière. Ils subdivisent régulièrement les groupes de 4 DDL : les cellules d'intégration à adapter sont subdivisées en 4 sous cellules (en

2D) et 5 nouveaux noeuds sont ajoutés. La figure 2.2 (tirée de [8]) illustre le procédé.

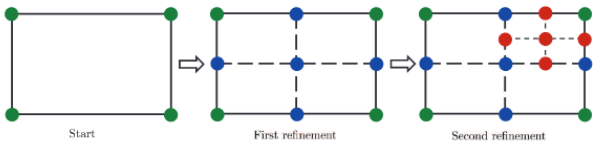

Figure 2.2 Subdivision régulière

Nous utiliserons une méthode de subdivision généralisant en 3D cette description 2D.

2.5.2 Subdivision régulière avec lissage

La stratégie de raffinement de Combe et al. [11] consiste en deux étapes :

Premièrement, la quantité $|u^h(\boldsymbol{x})|_{H^2(\Omega)}$ est évaluée dans chaque cellule d'intégration. Ensuite, les dimensions de la cellule sont utilisées pour déterminer h_I (longueur d'un côté de la cellule). Si le produit $h_I|u(\boldsymbol{x})|_{H^2(\Omega)}$ excède un seuil δ, la cellule d'intégration est divisée en 4 sous-cellules. Un noeud est ensuite ajouté en chaque centre des nouvelles cellules.

La seconde partie de la procédure peut être vue comme un post-traitement pour lisser les transitions entre les zones des différents noeuds. On vérifie qu'une cellule d'intégration n'a pas plus de deux voisines sur un de ses côtés. Si cela advient, on la subdivise. Plusieurs itérations sont généralement nécessaires pour avoir des transitions fluides sur l'ensemble du domaine.

On remarque que le lissage de Combe et al. [11] est très proche de celui adopté par Rabzuk et al. [13].

2.5.3 Subdivision avec critère de Devloo et Oden

Dans Rabzuk et al. [13], le domaine est subdivisé en pixels (voxels en 3D) qui coïncident avec les cellules d'intégration. Si l'erreur dépasse un certain critère, d'autres particules sont ajoutées. La figure 2.3 (tirée de [13]) illustre la procédure en 2D.

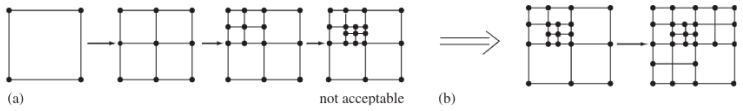

Figure 2.3 Adaptation du maillage 2D

On remarque que la règle de Devloo et Oden a été appliquée. Elle stipule que les approximations MLS se détériorent si la taille des voisinages de pixels diffère trop. Ainsi on subdivise les cellules voisines s'il y a plus de deux degrés de raffinement de différence.

La figure 2.4 (tirée de [13]) illustre le concept en 3D. Le critère de raffinement est quant à lui similaire à celui utilisé dans Combe et al. [11].

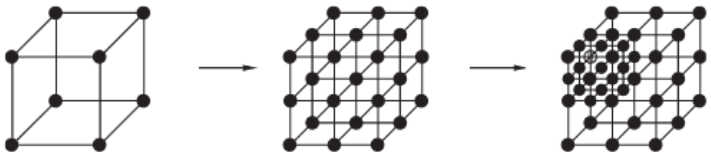

Figure 2.4 Adaptation du maillage 3D

La méthode de subdivision des DDL présentée par Rabzuk et al. [13] est identique à la notre. Nous avons également implémenté le critère de Devloo et Oden dans l'adaptation par seuils détaillée à la section 3.3.2.

2.5.4 Raffinement par diagramme de Voronoi

Yvonnet et al. [14] proposent de construire un diagramme de Voronoi à partir des noeuds du domaine. Le raffinement s'effectue en ajoutant des noeuds sur les arêtes des cellules de Voronoi dont l'erreur dépasse un certain seuil. Une fois ceci effectué, le diagramme est recalculé. Cette opération est répétée jusqu'à atteindre la tolérance voulue. La figure 2.5 (tirée de [14]) illustre l'opération.

Scyther utilise plutôt une grille régulière de DDL, qui, si elle est moins flexible, évite les calculs liés au raffinement du diagramme de Voronoi.

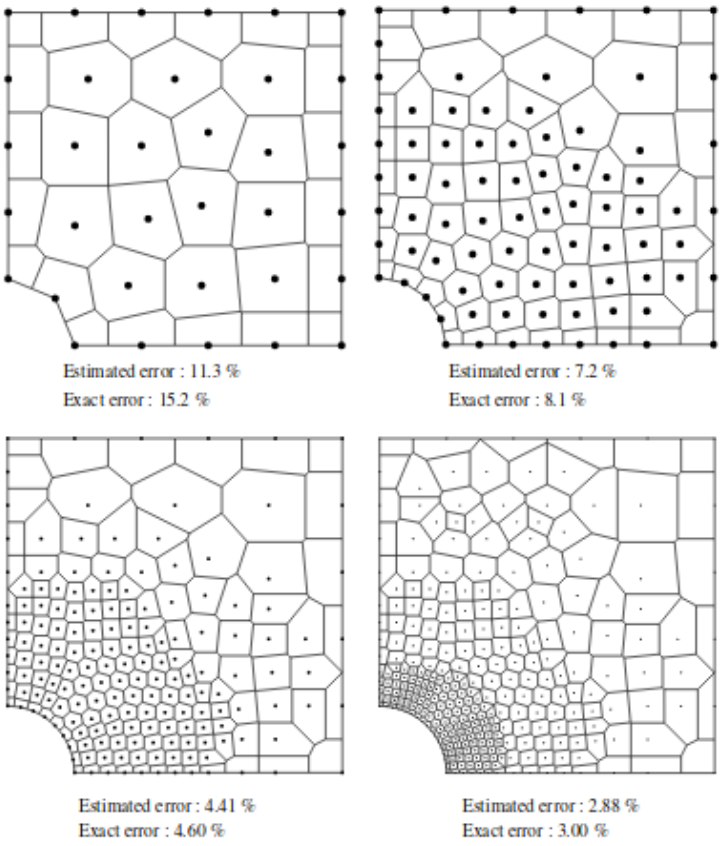

Figure 2.5 Raffinement du maillage par diagramme de Voronoi

2.5.5 Raffinement des cellules d'intégration

Joldes et al. [15] proposent une approche différente des autres articles cités précédemment. Il s'agit ici d'adapter le schéma d'intégration si l'intégrande est une fonction peu régulière.

Ils insistent sur le fait qu'augmenter le nombre de points de quadrature n'augmente pas nécessairement la précision du modèle, ce qui est le cas pour l'augmentation du nombre de cellules d'intégration. Ainsi, la technique proposée crée de nouvelles cellules d'intégration avec des poids associés à ces dernières.

On peut dresser un algorithme schématisant le procédé (avec une fonction f, un intervalle

d'intégration $[a, b]$ et une précision τ) comme présenté dans l'algorithme 1.

Algorithme 1 : Algorithme de raffinement des cellules d'intégration

```
Procédure intégrer(f, [a,b], τ) :
Q = quadrature(f, [a,b]);
ϵ = estimerErreur(f, [a,b], I);
si ϵ > τ alors
    m = (a+b)/2;
    Q = intégrer(f, [a,m], τ') + intégrer(f, [m,b], τ');
fin
retourner Q;
```

La résolution du système passe par la construction de la matrice de rigidité :

$$K_{ij} = \int_a^b \Phi_{i,x}(\boldsymbol{x})\Phi_{j,x}(\boldsymbol{x})dx \tag{2.58}$$

Joldes et al. [15] indiquent que les fonctions de base ont un support compact, ce qui implique que la plupart seront nulles sur une cellule d'intégration donnée. D'autre part, pour chaque point d'intégration d'une même cellule, il est possible que les fonctions de base non nulles soient différentes. Comme les fonctions de base sont combinées dans l'équation (2.58) pour former les éléments de la matrice de rigidité, beaucoup de fonctions différentes doivent être intégrées sur chaque cellule d'intégration. On pourrait calculer chaque intégrale en utilisant une procédure adaptative, mais cette approche serait trop gourmande en calculs.

Joldes et al. [15] stipulent l'hypothèse que la précision de l'intégration dépend de la régularité de la fonction. La régularité est définie au sens du degré minimal pour lequel un polynôme peut en faire l'approximation sur une cellule. Ainsi, si la méthode d'intégration adaptative est capable d'intégrer un intégrande donné, elle sera aussi capable d'intégrer efficacement toute fonction plus régulière. La méthode proposée comporte donc les étapes suivantes :

1. définir un intégrande moins régulier que toutes les fonctions utilisées dans le calcul de la matrice de rigidité de la cellule.
2. appliquer la procédure d'intégration adaptative sur cet intégrande à la précision voulue. Cela génère une collection de points d'intégration et de poids sur la cellule.
3. utiliser les points créés à l'étape précédente pour calculer la matrice de rigidité.

Il faut ensuite déterminer quelle fonction moins régulière utiliser :

— Si une fonction f peut être approchée par par un polynôme de degré n et qu'une fonction g est approchable par un polynôme de degré m, alors leur produit peut être approché par un polynôme de degré $n+m$. Comme $m+n < \max(2m, 2n)$, on peut utiliser le carré de la fonction la moins régulière pour avoir une fonction moins régulière que fg.
— La somme $f^2 + g^2$ peut être approchée par un polynôme de degré $\max(2m, 2n)$ (sauf dans le cas où les deux polynômes sont de même degré et que leurs coefficients du degré le plus élevé s'annulent).

En considérant les points précédents, on peut construire une fonction moins régulière permettant la construction des points d'intégration et des poids :

$$f(\boldsymbol{x}) = \sum_i (\Phi_{i,x})^2$$

On remarque que pour chaque point d'intégration, seules certaines fonctions de base sont non nulles. Par conséquent, la fonction précédente peut être facilement calculée en ces points.

Cette procédure produit une distribution de points d'intégration qui pour une précision donnée est définie uniquement par les fonctions de base. Ainsi, elle ne dépend que de la discrétisation du domaine (nombre et position des noeuds).

Cette méthode fonctionne bien tant que l'on ne veut pas ajouter des DDL ; sinon, les fonctions de forme changent et on doit refaire tous les calculs précédents. Cela semble empêcher une combinaison efficace de cette dernière et d'autres méthodes d'adaptation.

2.6 Transfert de données des noeuds

2.6.1 Transfert par interpolation

Ullah et al. [8] proposent de donner comme valeurs des champs aux nouveaux noeuds la pondération des anciens noeuds :

$$\xi(\mathbf{x}_{new}) = \sum_{i=1}^{n_{old}} \psi_i(\mathbf{x}_{new}) \xi(\mathbf{x}_{old})$$

où ξ est une grandeur quelconque (σ, ϵ ...). Le calcul des positions courantes est en particulier

calculé de cette manière. Cette manière de distribuer les grandeurs concorde avec la façon dont on généralise le déplacement en dehors des points échantillonnés.

2.6.2 Transfert par interpolation avec rayons variables

Rabzuk et al. [13] proposent d'adapter le rayon des nouveaux noeuds ajoutés. Les valeurs en chaque nouveau noeud sont toujours déterminées par interpolation avec les anciens noeuds.

Dans la configuration initiale, les particules sont espacées uniformément. Le rayon de support des fonctions de forme est un multiple de la distance entre ces dernières, usuellement $h \approx 3dx$, où dx est la distance entre particules. Si le raffinement est effectué, un nouveau rayon doit être assigné aux nouvelles particules et les anciennes dans la zone de transition. Si le domaine d'influence change trop brusquement, on risque d'avoir une mauvaise approximation. Pour créer des transitions plus lisses, le nouveau rayon de support est calculé par $h_k = 0.5(h_I + h_J)$ où K représente les particules affectées par le raffinement, h_I et h_J sont les rayons des particules représentées par la figure 2.6 (tirée de [13]).

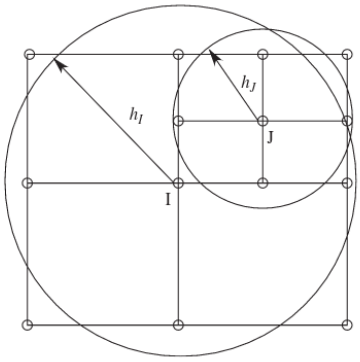

Figure 2.6 Adaptation avec rayon variable

La procédure de raffinement est illustrée par le schéma 2.7 (tirée de [13]).

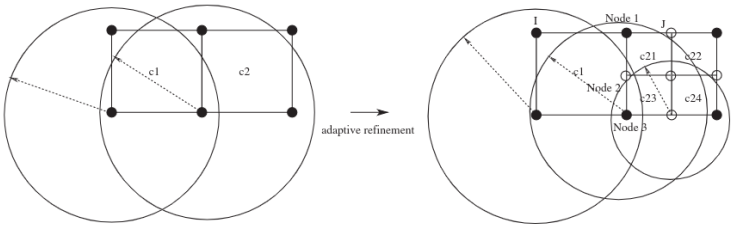

Figure 2.7 Schéma de la variation du rayon entre adaptations successives

Nous utilisons un système similaire d'adaptation des rayons à la subdivision, puisque nos fonctions de forme sont calculées selon un ensemble de DDL qui forment un voisinage de la cellule d'intégration. Leurs distances au PI de la cellule sont utilisées dans le calcul des fonctions de forme.

2.6.3 Adaptativité des rayons dans MLPG

Kamranian et al. [4] donnent un algorithme adaptatif pour la méthode MLPG B. Des noeuds peuvent être ajoutés ou retirés selon un critère lié à l'erreur. De plus, les rayons d'influence de ces noeuds dépendent de la distance avec leurs voisins. L'étape d'ajustement des rayons nécessite de déterminer les plus proches voisins d'un noeud, ce qui peut être une procédure lourde en calculs. Les kd-trees (k-dimensionnal trees) sont utilisés dans Kamranian et al. [4]. Ce sont des structures permettant de partitionner l'espace avec une complexité en $O(n \log n)$. Une fois l'arbre construit, une requête peut être effectuée en $O(\log n)$ ce qui est avantageux par rapport à $O(n^2)$ sans structure particulière.

Comme nous utiliserons une grille régulière de DDL nous n'aurons pas besoin de kd-trees. Cela nous économisera ces calculs, au détriment de la flexibilité du système.

CHAPITRE 3 MÉTHODOLOGIE

Ce chapitre explique le fonctionnement du simulateur. Il s'articule autour de :

- la topologie que nous avons choisie pour l'intégration du problème présenté dans la partie théorique ;
- l'initialisation des structures de données ;
- l'adaptation ;
- les tests et vérifications.

Dans un souci de concision, nous nous concentrerons sur les aspects du simulateur qui sont directement liés à l'adaptativité. De plus, nous identifierons les cellules d'intégration à leur PI de Gauss par métonymie.

3.1 Implémentation dans Scyther

La résolution du système linéaire par les gradients conjugués ainsi que l'affichage 3D est géré par Simulation Open Framework Architecture (SOFA), une plateforme pour la simulation physique. En conjonction avec SOFA, l'article [3] présente Scyther qui est la librairie réalisant les fonctionnalités du simulateur ainsi que l'assemblage des matrices. En outre, pour connecter SOFA et Scyther, il est nécessaire d'utiliser un plugin, qui a été nommé SofaScyther.

Le code de Scyther et de SofaScyther est écrit dans le langage C++. Le système d'adaptation ajouté dans le cadre de cette maîtrise est lui aussi en C++. La définition des scènes pour la simulation est en Python, tout comme les codes des analyses que nous présenterons dans nos résultats.

3.1.1 Topologie

Nous avons mentionné les volumes d'intégration lors de la discrétisation spatiale 2.2.2 qui divisent l'objet pour l'intégrer plus facilement dans la MGSM. Nous nous sommes jusqu'ici placés dans un cas général, sans discuter de la position ni des voisinages pour un élément Ω_k. Nous proposons de les définir ici.

La topologie utilisée dans Scyther est représentée à la figure 3.1. On nommera ce regroupement de cellules d'intégration et de DDL un octant. Cela nous sera utile par la suite lorsque

nous voudrons effectuer des adaptations. Dans un octant, chaque volume d'intégration Ω_k a comme voisinage V_k les 8 DDL aux sommets.

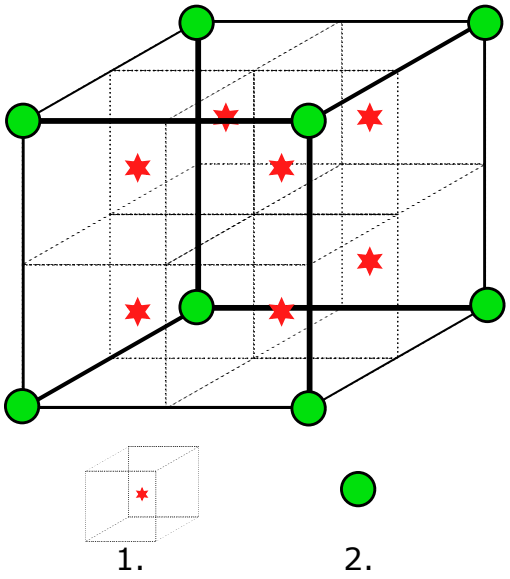

Figure 3.1 Topologie d'un octant avec 1. une cellule d'intégration et 2. un degré de liberté

L'assemblage des octants donne un objet. Un exemple illustratif est donné dans 3.2. On remarque que certains octants peuvent ne pas être complets, c'est-à-dire qu'ils peuvent posséder moins de 8 DDL et/ou moins de 8 PI.

3.1.2 Structures de données

Les *ObjectStates* sont les structures de données correspondant aux différentes caractéristiques physiques et géométriques de l'objet déformable.

- *DegreesOfFreedom* : les degrés de liberté du système, ayant pour chacun les attributs suivants : position à l'état initial, position courante et masse.
- *IntegrationPoints* : les points sur lesquels on effectue les quadratures de Gauss du premier ordre. On peut faire une correspondance directe entre ces points et les volumes d'intégration associés. Les membres sont : la position à l'état initial, la position courante, le volume, l'estimation de l'erreur.
- *OctreeGridStructure* : la structure de donnée découpant l'espace en octants. Il s'agit d'un octree dont les feuilles sont des octants.
- *MaterialProperties* : les propriétés du matériau de l'objet. Elles incluent le module de Young, le ratio de Poisson, la densité de masse et les coefficients de Lamé.

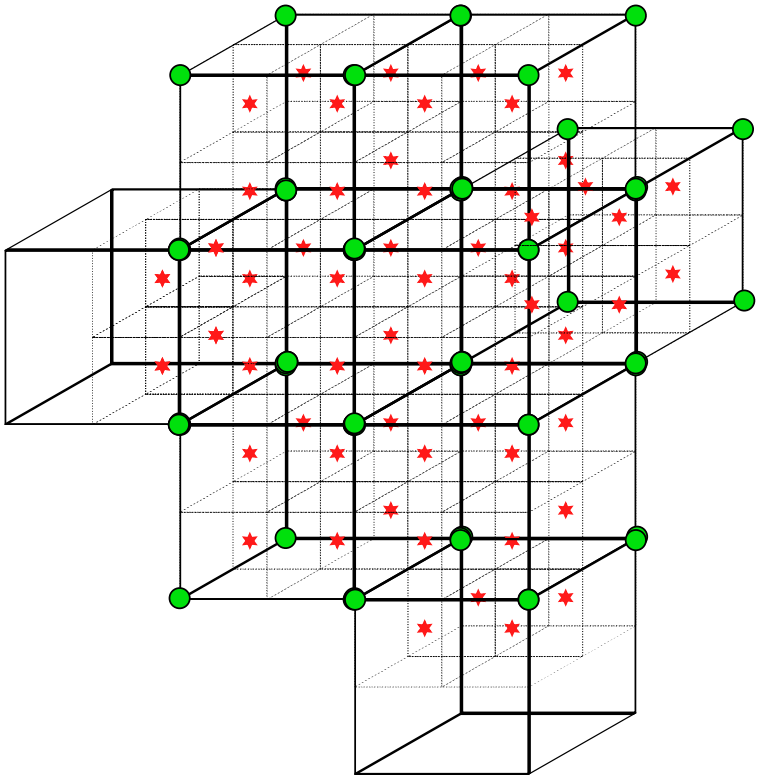

Figure 3.2 Exemple d'objet constitué de plusieurs octants

— *ParticleConnectivity* : les voisinages des PI (V_k), qui nous permettent de calculer les intégrales.
— *ShapeFunctionMapping* : contient les valeurs des fonctions de forme et de leurs dérivées pour chaque DDL voisin de chaque point d'intégration.
— *SurfaceMapping* : correspondance entre les DDL et les sommets de la surface (purement esthétique tant que l'on n'effectue pas de découpe, ce qui n'arrivera pas puisque nous avons spécifié que nous ne prenons pas en compte la découpe dans les limites de ce mémoire)

3.1.3 Acteurs

Les acteurs sont les classes gérant la modification ou la vérification des *ObjectStates*. Les objets de cette classe requièrent à leur création des références vers les *ObjectStates* qui sont utilisés en lecture et en écriture. Cela permet un meilleur cloisonnement des différentes actions à mener sur l'objet.

— *ConnectivityCalculator* : permet de calculer la connectivité lorsque l'on décime ou

subdivise un octant.
— *ShapeFunctionsChecker* : permet de vérifier la cohérence des fonctions de forme.
— *SurfaceMappingFullChecker* : permet de vérifier la cohérence de la correspondance DDL / sommets de la surface.
— *MassCalculator* : permet de calculer la masse d'un DDL selon sa connectivité
— *DofRemover* : permet de désactiver un DDL. Cet acteur met à jour toutes les structures de données impactées par cette action.
— *IntPointAdder* : crée de nouveaux PI en s'assurant que les autres structures de données restent cohérentes. Nécessite la position à l'état initial , les DDL selon lesquels on interpole la position courante (par les fonctions de forme), ainsi que le volume de la cellule d'intégration associée au PI.
— *IntPointRemover* : même chose que *DofRemover*, mais pour les PI.
— *ShapeCalculator* : calcule les fonctions de forme selon la connectivité.
— *SurfaceMapper* : permet de mapper les DDL aux sommets de la surface.

3.2 Initialisation

La description initiale de l'objet est composée de :

— sa surface,
— ses coefficients de Lamé,
— la dimension des volumes d'intégration.

Nous ferons les schémas de cette partie en 2D pour simplifier les diagrammes. Il est en effet impossible de représenter convenablement une surface fermée en 3 dimensions sur un support à 2 dimensions. L'équivalent 2D (une courbe fermée) est quant à lui plus clair et nous n'avons pas de spécificités supplémentaires en 3 dimensions en ce qui concerne l'initialisation.

On obtient à la première étape la topologie à la figure 3.3. On remarque que certains octants n'ont pas le nombre complet de cellules d'intégrations (4 en 2D, 8 en 3D) car le volume de l'objet qu'elles auraient représenté est trop faible.

Nous initialisons ensuite la structure d'octree. Pour cela, on trouve tout d'abord la dimension ayant le nombre maximal d'éléments puis on détermine la taille de l'octree $octreeSize$:

$$octreeSize = elemVolume \cdot \min\{n \in \mathbb{N} \mid (\exists p \in \mathbb{N}, n = 2^p) \wedge n \geq maxElem\} \tag{3.1}$$

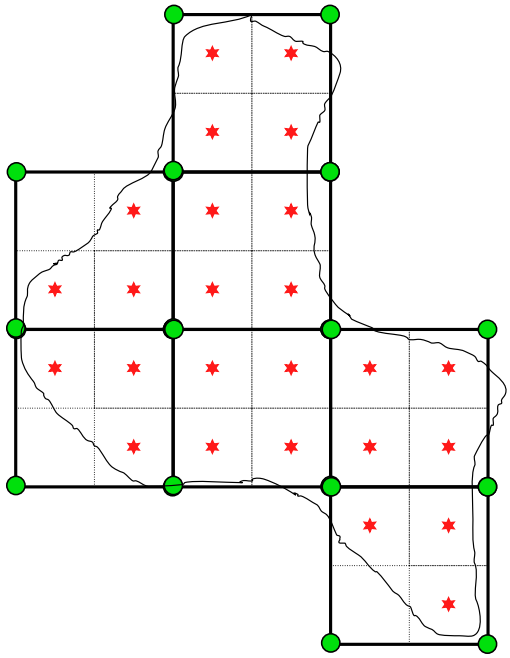

Figure 3.3 Objet au début de l'initialisation

Où $elemVolume$ est le volume d'intégration initial défini par l'utilisateur et $maxElem$ est le nombre maximal d'éléments sur un des axes (e_x ou e_y en 2D, auxquels s'ajoute e_z en 3D).

Ainsi, on a la taille de l'octree englobant tout l'objet. Avec la position des DDL et des PI, on peut alors construire récursivement l'octree en effectuant des tests sur la présence des DDL dans chaque sous-octant. La figure 3.4 nous montre deux exemples d'octree englobants. On remarque qu'il est possible que la plupart de l'octree soit vide ; cependant, seuls les octants effectivement remplis sont subdivisés.

La structure d'octree permet de calculer la connectivité simplement, en connectant dans chaque octant tous les PI aux DDL des sommets, ce qui forme pour la cellule d'intégration Ω_k son voisinage V_k (voir la partie théorique à la section 2.2). Nous illustrons cela par des liaisons bleues dans le schéma 3.5.

L'étape suivante est de supprimer les DDL ayant strictement moins de 5 PI voisins en 3D. La figure 3.6 illustre cela en 2D.

On finit par calculer les fonctions de forme $\phi_{ki}(x_k)$, les masses selon l'équation (2.34) et les *SurfaceMappings*.

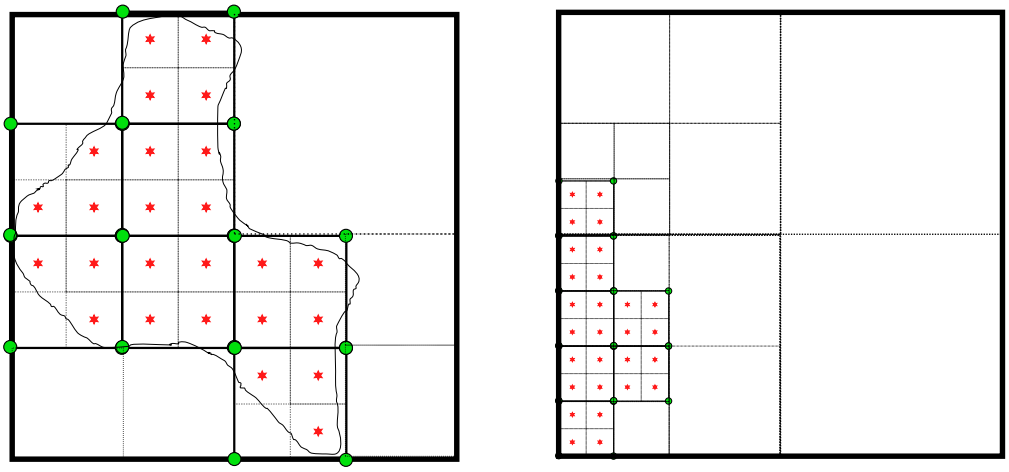

Figure 3.4 Octant englobant dans deux exemples

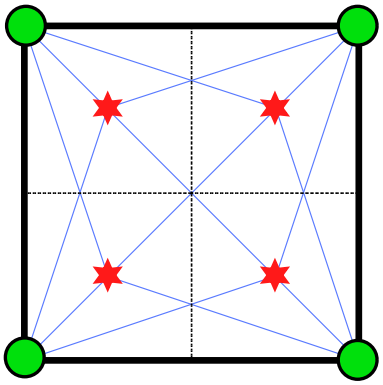

Figure 3.5 Voisinages dans l'octant

3.3 Adaptation

L'adaptation se déroule en plusieurs étapes. Nous devons commencer par estimer l'erreur pour ensuite décider selon des critères précis où l'on doit subdiviser ou décimer le système.

Formellement, soit o un octant, d un DDL, et i un PI, on pose :

Les PI	I
Les DDL	D
Feuilles de l'octree	F
Sous-octants enfants de o	$E(o)$
DDL de l'octant o	$D(o)$
PI de l'octant o	$I(o)$
Sommets de l'octant o	$S(o)$

Octants voisins sur les faces de l'octant o	$V_f(o)$
Octants voisins sur les arêtes de l'octant o	$V_a(o)$
DDL voisins du PI i	$V_{DDL}(i)$
PI voisins du DDL d	$V_{PI}(d)$
Prédicat vrai si o est une feuille de l'octree	$F(o)$
Parent de o	$P(o)$
Position à l'état initial associé au PI ou DDL p	$R(p)$
Position courante associée au PI ou DDL p	$C(p)$
Volume associé au PI i	$V(i)$
Masse associée au DDL d	$m(d)$
Estimé de l'erreur sur la cellule d'intégration du PI i	$e(i)$
Volume d'intégration associé au PI i	Ω_i
Nombre de DDL ajoutés si l'on subdivise o	$N_{sub}(o)$

3.3.1 Estimation de l'erreur

On fait la supposition que plus le gradient du déplacement est élevé sur un élément, plus il y a de chances que notre approximation soit éloignée de la véritable solution du problème à cause du manque de granularité des éléments. On pose donc la formule suivante comme notre estimateur d'erreur :

$$e(i) = \sqrt{\sum_{i=1}^{3}\sum_{j=1}^{3}((\nabla_x u)_{ij}(R(i)))^2} \cdot V(i) \tag{3.2}$$

où i est un PI.

Cette formule correspond à la norme de Frobenius du gradient du déplacement au PI i multiplié par le volume de Ω_i. Elle est simple à implémenter, pourvu qu'on dispose déjà des valeurs du gradient de u (ce qui est le cas dans la boucle d'assemblage des matrices).

Il faut se rappeler que nous n'utilisons pas réellement le gradient $\nabla_x u$, mais S, la partie non rotative de sa décomposition polaire.

3.3.2 Critères d'adaptation

Le critère d'adaptation est le mécanisme qui, à partir de l'estimé de l'erreur e, active la subdivision ou la décimation. À chaque itération du simulateur, on collecte les feuilles de

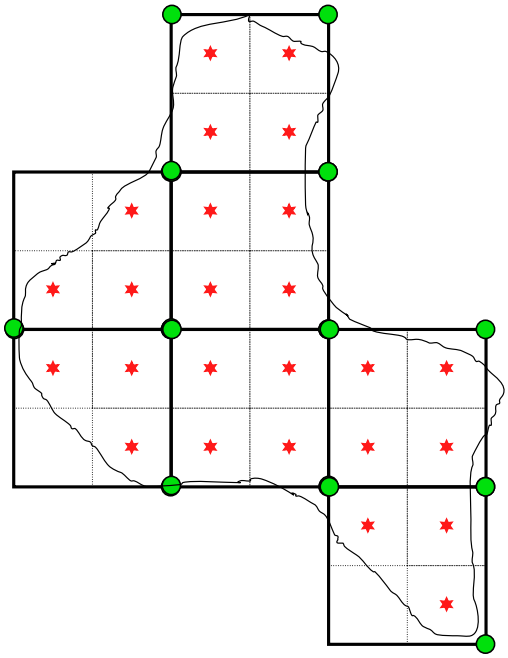

Figure 3.6 Objet en fin d'initialisation

l'octree, et on décide s'il faut les subdiviser ou non. Notre méthode d'adaptation se fait donc au niveau de l'octant, et non de la cellule d'intégration. En outre, précisons qu'on ne subdivise ou décime un octant qu'une seule fois par itération du simulateur.

Par seuils

On définit arbitrairement un premier seuil pour la subdivision, puis un second pour la décimation. On évite ainsi des problèmes d'oscillations avec une hystérésis.

Pour la subdivision, la condition est, pour o un octant des feuilles de l'octree :

$$\frac{\sum_{i \in I(o)} e(i)}{|I(o)|} > S_{sub} \ \wedge \ (|I(o)| = 8) \ \wedge \ (|D(o)| = 8) \tag{3.3}$$

Où S_{sub} est le seuil de subdivision fixé arbitrairement. On ne subdivise que des octants entièrement pleins, c'est-à-dire avec 8 DDL et 8 PI. Si cette condition est remplie, le mécanisme décrit à la partie Subdivision 3.3.3 est enclenché.

La décision de décimation n'est pas symétrique avec celle de la subdivision : en effet, nous subdivisons un octant, alors que nous décimons un octant déjà subdivisé contenant des octants non subdivisés. Cela se traduit mathématiquement par :

$$(\forall c \in E(P(o)), F(c)) \ \wedge \ (\forall c \in E(P(o)), \frac{\sum_{i \in I(c)} e(i)}{|I(c)|} < S_{dec}) \tag{3.4}$$

Où S_{dec} est le seuil de décimation choisi arbitrairement, mais plus petit que S_{sub}. Une fois cette condition remplie, on active la décimation sur o présentée à la section Décimation 3.3.4. On ne considère que les octants déjà subdivisés ; il est donc impossible d'avoir moins de DDL qu'à l'état initial.

On remarque que les octants à décimer ne sont pas des feuilles de l'octree mais sont situés au niveau juste au-dessus. On les collecte avec la prévérification suivante sur les feuilles :

$$\frac{\sum_{i \in I(c)} e(i)}{|I(c)|} < S_{dec} \tag{3.5}$$

qui est une condition nécessaire sur les enfants d'un octant à décimer. Si cette dernière est validée, on vérifie si $P(c)$ vérifie le prédicat 3.4.

En outre nous implémentons la règle de Devloo et Oden présentée dans la revue de littérature à la section 2.5.3 avec ce critère d'adaptation.

Par erreur relative

Le critère d'adaptation par seuils possède une faiblesse majeure : on doit fixer arbitrairement leurs valeurs. Des valeurs effectives pour ces seuils vont extrêmement varier selon la topologie et les forces du problème. De plus, dans une situation courante d'usage du simulateur, on ne s'attend pas à ce que l'utilisateur fournisse ces paramètres. D'autre part, il est impossible d'avoir des garanties sur le nombre de DDL utilisés, on ne peut spécifier qu'un nombre maximal à ne pas dépasser.

Ces raisons font que nous avons développé un système indépendant de la valeur de l'erreur.

Posons l'erreur sur un octant o :

$$e(o) = \sum_{i \in I(o)} e(i) \tag{3.6}$$

Posons la moyenne de l'estimé de l'erreur sur les feuilles :

$$E_F = \frac{\sum_{o \in F} e(o)}{|F|} \tag{3.7}$$

Soit ϵ un réel quelconque (on prendra en pratique $\frac{E_F}{10}$) et m le nombre maximal de DDL que l'on souhaite ne pas dépasser. La condition de subdivision pour l'octant o est la suivante :

$$(e(o) > E_F + \epsilon) \ \wedge \ (|D| + N_{sub}(o) < m) \ \wedge \ (|I(o)| = 8) \ \wedge \ (|D(o)| = 8) \tag{3.8}$$

Où $N_{sub}(o)$ est le nombre de DDL ajoutés par la subdivision de o. En effet, ce nombre dépend des voisins déjà subdivisés de o, qui auront donc des DDL en commun avec les enfants de o si o est subdivisé. Le ϵ est utilisé de manière à créer une hystérésis prévenant l'oscillation entre deux étapes d'adaptation. Malgré cela, il peut demeurer des oscillations, surtout pour des forces externes soudaines et importantes.

On ne subdivise que des octants entièrement pleins, c'est-à-dire avec 8 DDL et 8 PI.

Pour la décimation, la condition de subdivision est la suivante :

$$(\forall c \in E(P(o)), F(c)) \ \wedge \ (e(P(o)) < 8(E_F - \epsilon)) \tag{3.9}$$

Où $e(P(o)) = \sum_{i \in I(P(o))} e(i)$.

De plus, pour utiliser le plus possible de DDL avant d'atteindre la limite maxDDL, on réalise une boucle de subdivision supplémentaire sur les feuilles de l'octree qui n'ont pas été décimées ou subdivisées à l'étape courante. La condition dans ce cas est toujours celle énoncée en 3.8.

La méthode que nous venons de présenter possède les caractéristiques suivantes :

— à un état de repos, on utilisera au moins $m - (27 - 8) + 1 = m - 18$ DDL ;
— on effectue au maximum $|F|$ décimations ou subdivisions, ce qui borne raisonnablement le temps de calcul associé à cette étape. On pourrait éventuellement mettre une limite stricte sur le nombre d'adaptations pour garantir des critères de performances en temps réel.

D'autres méthodes sont possibles, notamment celles qui effectuent des adaptations sur plusieurs niveaux d'affilée (comme n subdivisions par exemple). Elles rendent cependant l'adaptation plus gourmande en ressources, mais assurent qu'à tout instant on utilise le maximum de DDL et qu'on les a bien répartis.

3.3.3 Subdivision

Nous détaillons ici les différentes étapes de la subdivision dans l'ordre où elles sont effectuées. Nous notons o l'octant à subdiviser. La figure 3.7 illustre le procédé.

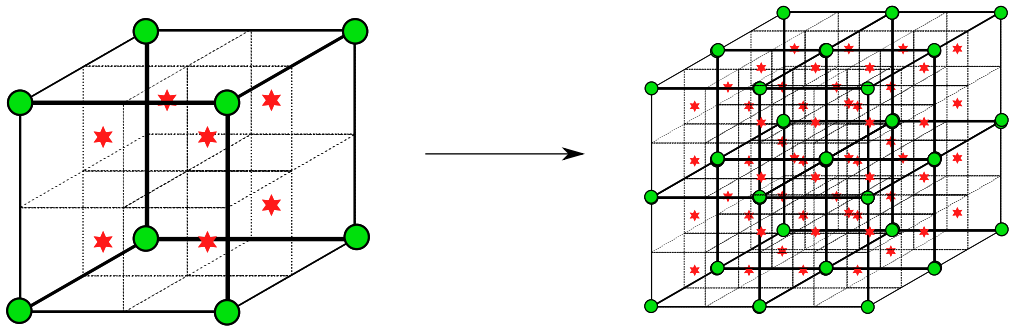

Figure 3.7 Processus de subdivision

Avant toute chose, nous devons utiliser une convention pour nous repérer au niveau des sommets de l'octant, mais aussi de ses faces et de ses arêtes. La figure 3.8 illustre la convention des sommets, des faces et des arêtes. On doit par ailleurs aussi se repérer dans le cube subdivisé, ce qui nous impose une autre convention à 27 noeuds. Cette dernière est présentée à la figure 3.9.

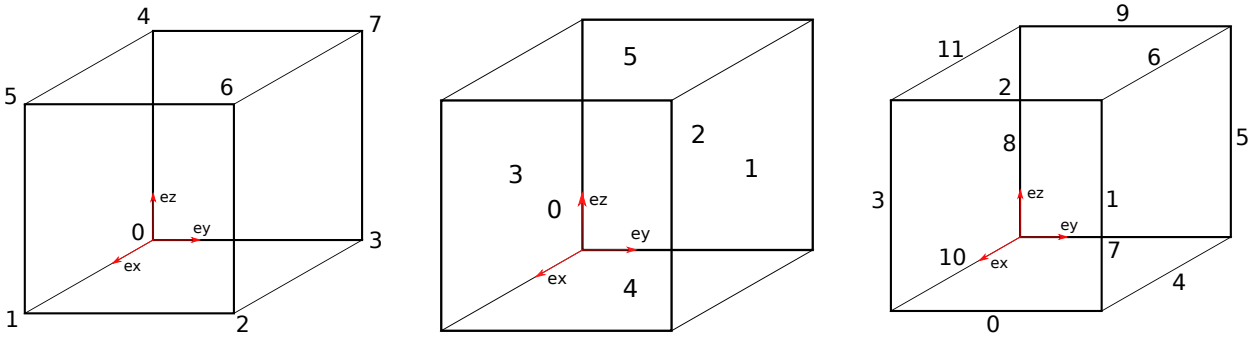

Figure 3.8 Convention de numérotation des sommets, des faces et des arêtes

Création des PI

Les trois membres que nous devons générer lors de la création de nouveaux PI sont la position à l'état initial , la position courante et le volume.

Pour la position à l'état initial , soit (e_x, e_y, e_z) la base canonique de $\mathbb{R}^3$. Soit c^{min} le coin ayant les plus petites coordonnées de tous les coins de l'octant o. On définit $c = c_{min} + \frac{h}{8}(e_x + e_y + e_z)$

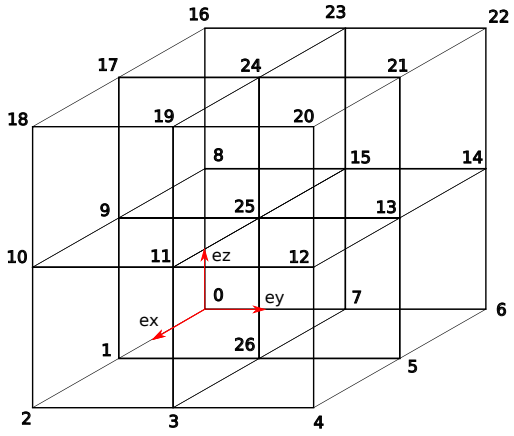

Figure 3.9 Convention de numérotation des DDL pour la subdivision

pour simplifier les calculs suivants. Soit h la dimension de l'octant. Comme notre octant sera subdivisé en 8 sous-octants, nous définissons c_i où i est le numéro de sommet. On a :

$$c_0 = c \tag{3.10}$$

$$c_1 = c + e_x \cdot \frac{h}{2} \tag{3.11}$$

$$c_2 = c + e_x \cdot \frac{h}{2} + e_y \cdot \frac{h}{2} \tag{3.12}$$

$$c_3 = c + e_y \cdot \frac{h}{2} \tag{3.13}$$

$$c_4 = c + e_z \cdot \frac{h}{2} \tag{3.14}$$

$$c_5 = c + e_x \cdot \frac{h}{2} + e_z \cdot \frac{h}{2} \tag{3.15}$$

$$c_6 = c + e_x \cdot \frac{h}{2} + e_y \cdot \frac{h}{2} + e_z \cdot \frac{h}{2} \tag{3.16}$$

$$c_7 = c + e_y \cdot \frac{h}{2} + e_z \cdot \frac{h}{2} \tag{3.17}$$

Ensuite, pour obtenir la position de chaque nouveau PI dans chaque sous-octant, on sélectionne le c_j associé à la bonne arête. Notons $(i_k)_{k\in[[1,8]]}$ les PI de ce sous-octant. On a leur position par les calculs suivants :

$$R(i_0) = c_j \tag{3.18}$$
$$R(i_1) = c_j + e_x \cdot \frac{h}{4} \tag{3.19}$$
$$R(i_2) = c_j + e_x \cdot \frac{h}{4} + e_y \cdot \frac{h}{4} \tag{3.20}$$
$$R(i_3) = c_j + e_y \cdot \frac{h}{4} \tag{3.21}$$
$$R(i_4) = c_j + e_z \cdot \frac{h}{4} \tag{3.22}$$
$$R(i_5) = c_j + e_x \cdot \frac{h}{4} + e_z \cdot \frac{h}{4} \tag{3.23}$$
$$R(i_6) = c_j + e_x \cdot \frac{h}{4} + e_y \cdot \frac{h}{4} + e_z \cdot \frac{h}{4} \tag{3.24}$$
$$R(i_7) = c_j + e_y \cdot \frac{h}{4} + e_z \cdot \frac{h}{4} \tag{3.25}$$

Pour la position courante, on utilise les fonctions de forme dans Ω_i où $i \in \mathrm{PI}(o)$:

$$C(i) = \sum_{d \in D(o)} \phi_{id}(R(i)) \cdot C(d) \tag{3.26}$$

Pour le volume, soit Ω_k un des 8 anciens éléments de o et soit les 8 nouveaux éléments $\Omega_j \subset \Omega_k$:

$$\forall s \in S(o), \forall i \in I(E(o)(s)), V(i) = \frac{V(I(o)(s))}{|S(o)|} \tag{3.27}$$

Expliquons les différents termes de cette équation :

— $E(o)(s)$ correspond à l'octant enfant de o au sommet s tel que spécifié par la convention 3.8 ;
— $I(o)(s)$ correspond au PI de o situé au sommet s de son cube interne de PI tel que spécifié par la convention 3.8.

En d'autres termes, on distribue le volume des cellules d'intégrations de manière à ce que les cellules descendantes aient $\frac{1}{8}$ du volume de la cellule originelle.

Il faut ensuite garder ces PI dans une structure de données adaptée pour les passer aux octants enfants. Nous avons utilisé un tableau de 8 tableaux contenant chacun les 8 index

des PI ordonnés selon la numérotation de la convention des sommets 3.8.

Création des DDL

La création des DDL à la subdivision est plus complexe que pour les PI. La raison est que des DDL peuvent déjà avoir été créés dans les octants voisins lors de subdivisions antérieures, et qu'il ne faut donc pas les dupliquer. La solution à ce problème a été de faire des connexions entre octants voisins. On différenciera les octants voisins par les faces et par les arêtes.

Les différentes classes de cas possibles sont présentées par les figures 3.10 et 3.11. On représente respectivement 1 cas sur 6 pour les faces et 1 cas sur 12 pour les arêtes.

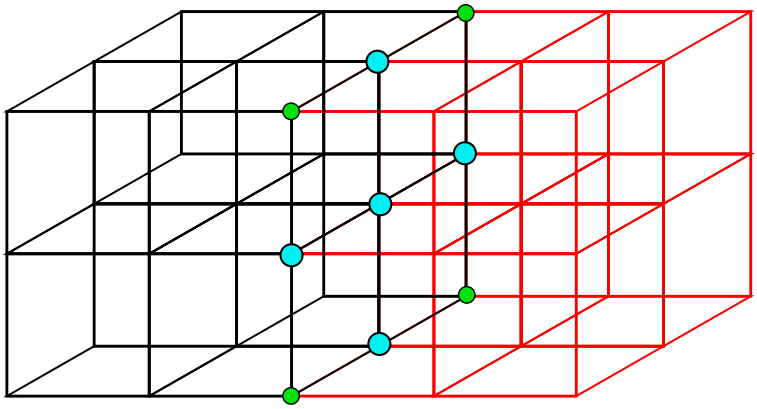

Figure 3.10 DDL en commun avec un voisin sur une face (en cyan, hors DDL déjà présents avant la subdivision)

On créera de nouveaux DDL que dans le cas où des DDL ne sont pas déjà présents aux emplacements en cyan des figures précédentes. Ainsi, les DDL de l'octant à subdiviser sont toujours conservés (mais leur masse changera).

Un système de tableaux d'adressage permet de faire la correspondance avec les DDL déjà existants dans les octants voisins.

Mise à jour des octants voisins

Pour pouvoir subdiviser ou décimer par la suite, il convient de garder la structure de l'octree conforme, c'est-à-dire que les voisins doivent être accessibles. Lorsque l'on subdivise un octant, la recherche des nouveaux octants voisins chez les enfants se décompose en deux :

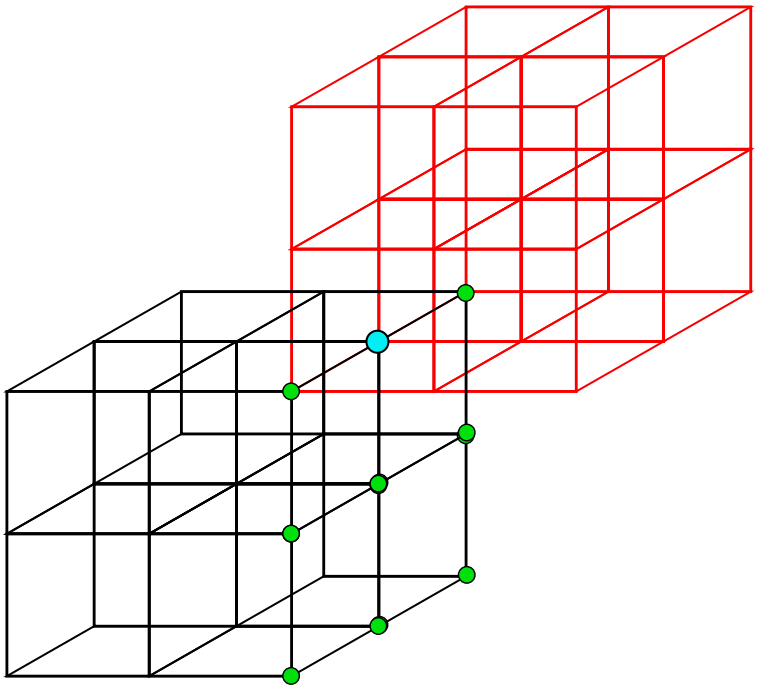

Figure 3.11 DDL en commun avec un voisin sur une arête (en cyan, hors DDL déjà présents avant la subdivision)

— les voisins contenus dans le parent, qui sont les autres octants qui viennent d'être créés. Il est aisé de les trouver par adressage.
— les voisins contenus dans les octants voisins du parent qui ont déjà été subdivisés. Dans ce cas, il faut naviguer dans l'octree et vérifier si les voisins sont déjà subdivisés.

On différencie également les voisins de faces et d'arêtes, mais les procédures sont très similaires. On donne des schémas d'exemples de ces cas aux figures 3.12 et 3.13.

Suppression des anciens PI

On doit ensuite supprimer les PI de l'octant subdivisé en s'assurant de garder les structures de données cohérentes. On modifie donc $SurfaceMappings$, $ParticleConnectivity$, $IntegrationPoints$ et $ShapeFunctions$. De plus, on garde en mémoire tous les DDL qui étaient voisins des PI supprimés car leurs fonctions de forme ont changé avec la disparition des PI.

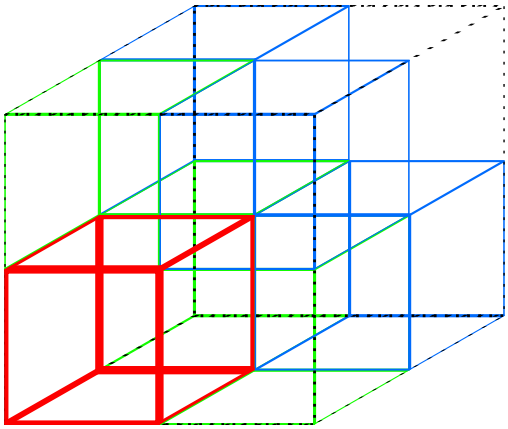

Figure 3.12 Voisins de l'octant rouge dans son parent : en vert les voisins sur ses faces, en bleu ceux sur ses arêtes, en pointillés le parent

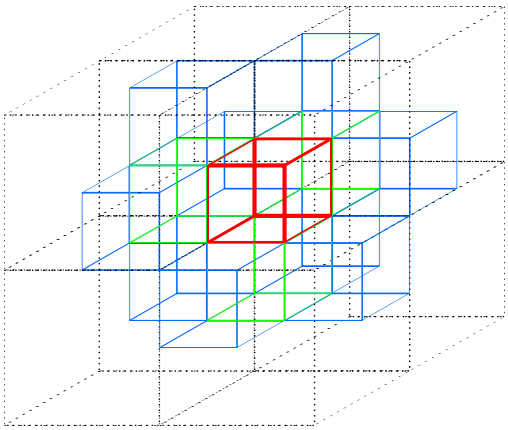

Figure 3.13 Voisins de l'octant rouge dans les voisins de son parent : en vert les voisins sur ses faces, en bleu ceux sur ses arêtes, en pointillés les octants parents

Mise à jour de la connectivité

La mise à jour de la connectivité est un point crucial de la subdivision est peut être réalisée de différentes manières.

Nous commençons par connecter tous les PI à tous les DDL dans chaque sous-octant. Ensuite, nous cherchons lesquels des octants voisins des faces sont situés au niveau de subdivision inférieur au nôtre (c'est-à-dire que nous sommes subdivisés une fois de plus). Les octants vérifiant cette condition voient leurs PI reliés aux DDL de notre octant subdivisé situés sur leur face. On représente cela dans la figure 3.14 : les liens de voisinage sont en pointillés. On n'y représente que les nouveaux liens.

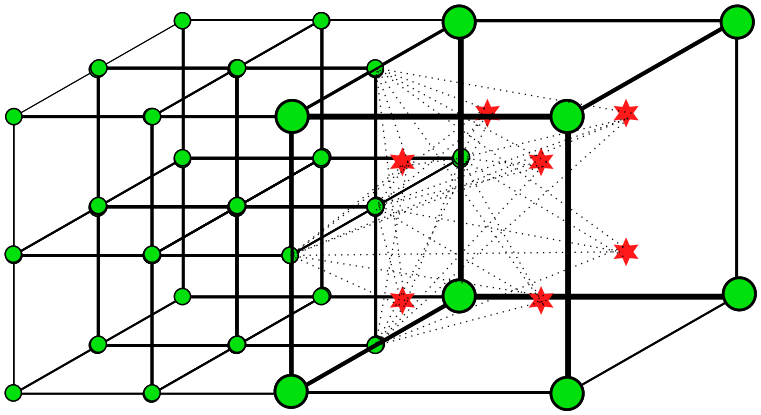

Figure 3.14 Connectivité entre octants de niveaux d'adaptation différents

Cette méthode présente au moins deux limites :

— on ne lie pas les PI des octants voisins à plus d'un degré de subdivision de distance.
— on ne prend pas en compte les voisins sur les arêtes.

Notons qu'il existe un réseau reliant les DDL entre eux, mais qui n'est utilisé que dans la partie du simulateur qui gère les découpes. Nous avons malgré tout mis à jour ces liens DDL-DDL pour permettre, à plus long terme, de relier notre module d'adaptation et le module de découpe.

Mise à jour des fonctions de forme

La modification des voisinages entre DDL et PI entraine le changement des fonctions de forme car elles sont dépendantes de l'ensemble des voisins pour chaque PI comme nous pouvons le voir dans leur formulation 2.9. Nous recalculons donc les fonctions de forme pour tous les PI ayant reçu des modifications dans leurs voisinages selon leur nouvelle connectivité avec la formule 2.17 énoncée dans la partie théorique.

Mise à jour des *SurfaceMappings*

La mise à jour des correspondances sur la surface n'impacte pas physiquement le système, donc nous ne faisons que mentionner l'existence de cette procédure.

Génération des masses

La génération des masses des DDL nécessite que la connectivité soit à jour. C'est pourquoi nous réalisons cette étape en fin de subdivision. On utilise la formule :

$$m(d) = \sum_{i \in V_{PI}(d)} \phi_{id}(R(i)) \cdot \rho \cdot V(i) \tag{3.28}$$

qui est tirée de l'équation (2.34).

3.3.4 Décimation

La décimation réutilise les conventions posées lors de la subdivision pour numéroter les 27 DDL présents dans un octant avant sa décimation. On illustre la procédure dans la figure 3.15.

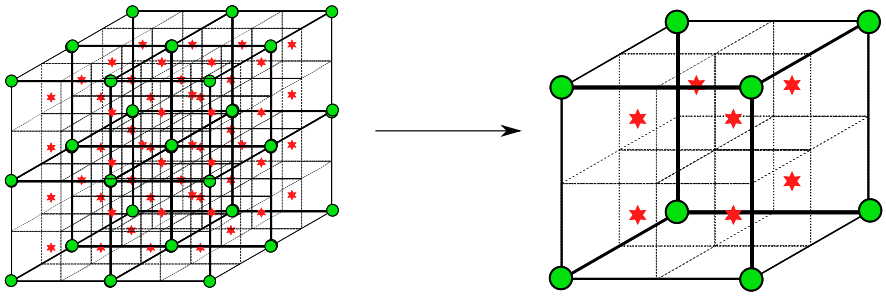

Figure 3.15 Processus de décimation

Suppression des DDL

De manière analogue à la subdivision, on ne veut pas supprimer des DDL qui seraient partagés avec d'autres octants voisins déjà subdivisés. Par exemple, les DDL aux sommets de l'octant à décimer sont toujours conservés. On présente cela à la figure 3.16 : les DDL verts sont toujours conservés, ceux en orange le sont conditionnellement, et celui en rouge jamais.

Gestion des PI

Comme pour la subdivision, on doit passer les données de position à l'état initial , de position courante, et de volume entre les anciens et les nouveaux points d'intégration.

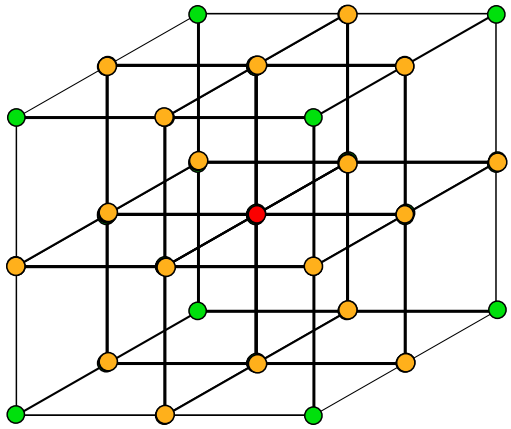

Figure 3.16 Octant juste avant une décimation, avec colorisation des DDL selon leur statut

Pour les positions à l'état initial , posons i_j les nouveaux PI, c_min l'arête de coordonnées minimales de l'octant, et $c = c_{min} + \frac{h}{8}(e_x + e_y + e_z)$. On calcule ensuite :

$$R(i_0) = c \tag{3.29}$$
$$R(i_1) = c + e_x \cdot \frac{h}{2} \tag{3.30}$$
$$R(i_2) = c + e_x \cdot \frac{h}{2} + e_y \cdot \frac{h}{2} \tag{3.31}$$
$$R(i_3) = c + e_y \cdot \frac{h}{2} \tag{3.32}$$
$$R(i_4) = c + e_z \cdot \frac{h}{2} \tag{3.33}$$
$$R(i_5) = c + e_x \cdot \frac{h}{2} + e_z \cdot \frac{h}{2} \tag{3.34}$$
$$R(i_6) = c + e_x \cdot \frac{h}{2} + e_y \cdot \frac{h}{2} + e_z \cdot \frac{h}{2} \tag{3.35}$$
$$R(i_7) = c + e_y \cdot \frac{h}{2} + e_z \cdot \frac{h}{2} \tag{3.36}$$

Pour les positions courantes, on utilise la même formule que pour la subdivision :

$$C(i) = \sum_{d \in D(o)} \phi_{id}(R(i)) \cdot C(d) \tag{3.37}$$

Pour le volume, posons $(\Omega_j)_{j \in J}$ les anciens éléments inclus dans Ω_k le nouvel élément. La formule est la suivante :

$$V(k) = \sum_{j \in J} V(j) \tag{3.38}$$

Mise à jour des octants voisins

Soit o l'octant que l'on décime.

On supprime les références aux octants enfants de o dans les enfants de $V_f(o)$ et $V_a(o)$ s'ils sont eux-mêmes subdivisés.

Mise à jour de la connectivité

On applique la même procédure que pour la subdivision, mais sur l'octant décimé cette fois.

Mise à jour des fonctions de forme

Comme pour la subdivision, la modification des voisinages entre DDL et PI entraine le changement des fonctions de forme car elles sont dépendantes de l'ensemble des voisins. Nous recalculons donc les fonctions de forme pour tous les PI ayant reçu des modifications dans leurs voisinages de la même manière que pour la subdivision.

Mise à jour des *SurfaceMappings*

Encore une fois, on mentionne ici la mise à jour des connexions entre les sommets de la surface et les DDL, mais nous ne la détaillerons pas car elle n'influe pas sur le comportement du système.

Génération des masses

La génération des masses s'effectue de la même manière que pour la subdivision.

3.4 Tests et vérifications

Pour nous assurer de la validité de notre code et pour trouver les éventuels bugs à résoudre avec leur contexte, nous avons écrit des scénarios de test sur différentes parties du programme.

Ils utilisent le framework *Google Tests*[1].

L'objet utilisé lors des tests est un cube paramétré de telle sorte qu'il contienne 8 octants. On peut le voir à la figure 3.17.

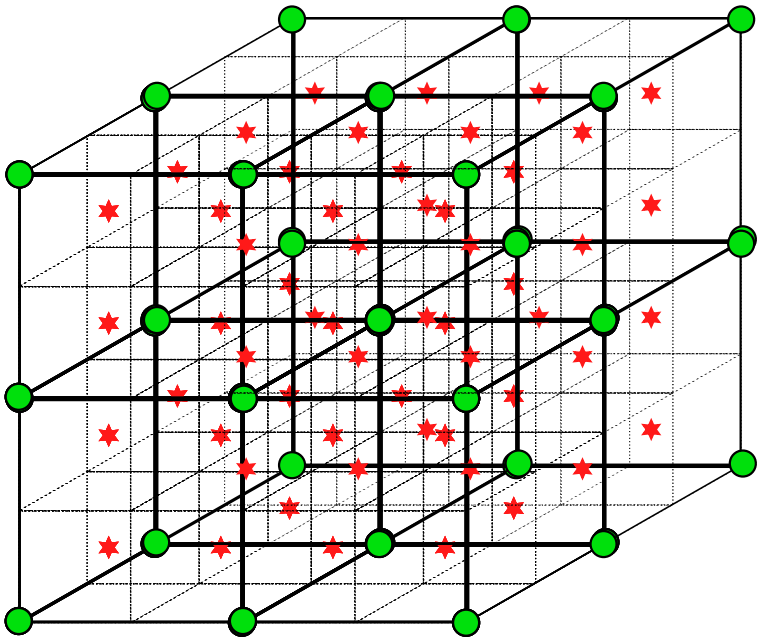

Figure 3.17 Cube de test

Pour vérifier la cohérence des structures de données, il est nécessaire de créer des acteurs spécifiques croisant les différentes structures entre elles. L'acteur *ShapeFunctionsChecker* en fait par exemple partie. De plus, chaque *ObjectState* contient des fonctions internes de vérification.

Nous avons écrit des tests déterministes en essayant de prendre en compte le plus de cas possible :

- un test exhaustif de l'ensemble des subdivisions au premier degré dans tous les ordres possibles. Cela correspond à $8! = 40320$ possibilités.
- un test se concentrant sur deux octants face à face et effectuant toutes les secondes subdivisions possibles dans ces deux octants. On a dans ce cas $8! = 40320$ possibilités.
- un test comme le précédent, mais sur les arêtes. On a $6! = 720$ cas.
- un test subdivisant et décimant plusieurs octants avant de revenir à l'état initial. Il vérifie ensuite si les structures de données sont les mêmes qu'à l'état de base.

1. https ://github.com/google/googletest

Comme les tests déterministes ne sont qu'une petite partie des cas possibles, nous avons jugé judicieux d'effectuer des tests aléatoires pour couvrir plus de possibilités. Leur fonctionnement repose sur une sélection aléatoire d'une feuille de l'octant, suivi aléatoirement soit d'une subdivision soit d'une décimation [2]. Ainsi, deux tests aléatoires ont été menés, un sur le cube précédent, et un dernier sur un tore pour être plus proche d'une condition d'utilisation classique. En effet, le tore possède des octants non pleins, ce qui peut potentiellement amener des bugs supplémentaires.

2. Nous ajoutons quelques critères supplémentaires : que l'octant soit plein (8 DDL et 8 PI) ; et, pour la décimation, que le parent ait des enfants qui sont tous des feuilles. De plus, nous imposons un niveau de profondeur de subdivision maximal.

CHAPITRE 4 RÉSULTATS et DISCUSSION

Dans cette partie, nous proposons de tester l'efficacité de notre méthode d'adaptation dans deux scénarios différents. On s'intéressera en premier à l'analyse quantitative du coût en temps de calcul et du gain en précision du module d'adaptativité. On étudiera ensuite le processus d'adaptation dans une analyse qualitative. Nous n'avons analysé que l'adaptation relative car c'est la seule méthode qui permet de fixer un nombre de DDL.

4.1 Analyse quantitative

4.1.1 Solution analytique

Nous utiliserons une poutre soumise à la gravité pour comparer le système avec et sans adaptation. Ce choix est guidé par l'existence et la simplicité des solutions analytiques.

On présente un schéma du problème à la figure 4.1.

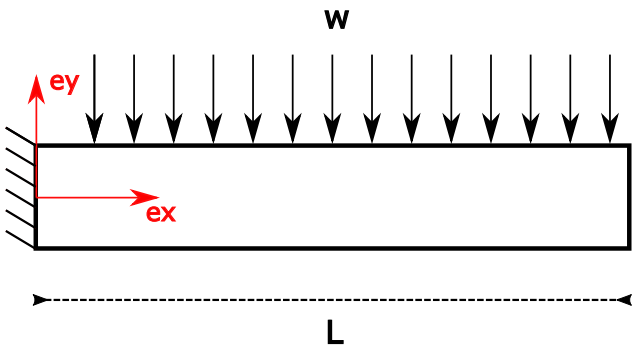

Figure 4.1 Poutre avec champ de force

Soit L la longueur de la poutre, w la densité de force linéique appliquée sur l'axe e_x, E le module de Young et I le moment quadratique. La formule de la déformation sur l'axe moyen est donnée par :

$$u_m = \frac{-wx^2}{24EI}(x^2 - 4Lx + 6L^2) \tag{4.1}$$

Et sa dérivée est :

$$u'_m = \frac{-wx}{6EI}(x^2 - 3Lx + 3L^2) \tag{4.2}$$

Nous utilisons une poutre carrée de dimension h . Son moment quadratique sur l'axe e_x est :

$$I = \frac{h^4}{12} \tag{4.3}$$

Les formes analytiques ne sont valables que pour l'axe moyen de la poutre. On doit donc encore trouver la déformation en tout point.

Pour l'axe e_z, on fait l'hypothèse de l'invariance au vu de la symétrie du problème. Pour e_y et e_x, on considérera que l'on est en petites déformations (hypothèse de Bernouilli), ainsi les sections droites restent planes et perpendiculaires à la courbe moyenne. Ceci est illustré à la figure 4.2 [1].

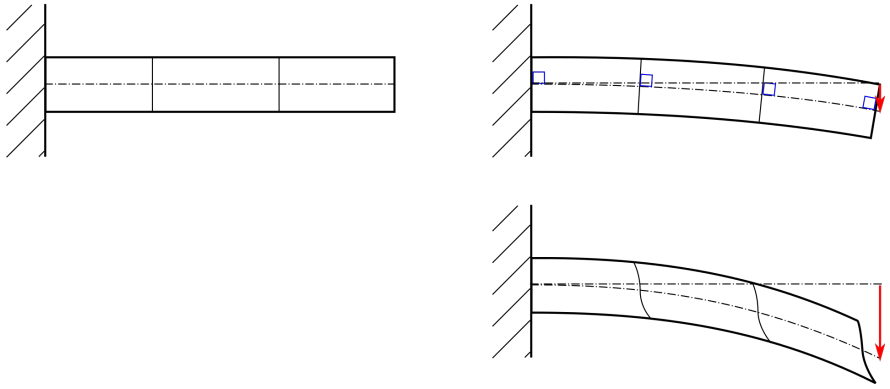

Figure 4.2 Hypothèse des petites déformations et des grandes déformations

On décompose la déformation en une rotation puis une translation. En un point $(x, 0)$ donné, l'angle θ de la rotation est lié au coefficient directeur de la tangente à la poutre déformée par la formule suivante :

$$\tan\theta = u'_m \tag{4.4}$$

On place notre repère en $(x, 0)$, puis on applique la rotation suivante sur le point $(0, y)$:

1. https://fr.wikipedia.org/wiki/Th%C3%A9orie_des_poutres#/media/Fichier:Poutre_hypotheses_deformation.svg

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} 0 \\ y \end{pmatrix} = \begin{pmatrix} -y\sin\theta \\ y\cos\theta \end{pmatrix} \tag{4.5}$$

En utilisant les formules de trigonométries suivantes :

$$\sin\ \arctan(x) = \frac{x}{\sqrt{1+x^2}} \tag{4.6}$$

$$\cos\ \arctan(x) = \frac{1}{\sqrt{1+x^2}} \tag{4.7}$$

Et en replaçant le repère à l'origine, tout en appliquant la déformation de l'axe moyen, on obtient :

$$u(x,y,z) = \begin{pmatrix} \frac{-u'_m}{\sqrt{1+u_m^2}}y \\ u_m + \frac{1}{\sqrt{1+u_m^2}}y - y \\ 0 \end{pmatrix} \tag{4.8}$$

En combinant cette équation avec la déflexion sur l'axe moyen, on détermine la déformation en tout point de la poutre.

4.1.2 Calcul de l'erreur

Il nous reste maintenant à déterminer l'erreur. Pour la mesurer, on calcule la norme L^2 de la différence entre la solution analytique u et de notre approximation u_h comme suit :

$$\|u - u_h\|_{L^2} = \sqrt{\int_\Omega \|u(x) - u_h(x)\|_2^2 dV} \tag{4.9}$$

On utilise les points d'intégration I de notre système avec une intégration de Gauss à 1 point, ce qui nous donne :

$$\|u - u_h\|_{L^2} \approx \sqrt{\sum_{i\in I} \|u(x_i) - u_h(x_i)\|_2^2 \cdot vol(i)} \tag{4.10}$$

Où $x_i = R(i)$.

Le problème est ici que cette quantité doit être exprimée dans des unités dépendantes des unités des variables du problème. On peut remédier à cela en optant pour la mesure d'erreur relative suivante :

$$E_r = \frac{\|u - u_h\|_{L^2}}{\|u\|_{L^2}} \tag{4.11}$$

Que l'on calcule en pratique avec des quadratures de Gauss :

$$E_r \approx \sqrt{\frac{\sum_{i \in I} \|u(x_i) - u_h(x_i)\|_2^2 \cdot vol(i)}{\sum_{i \in I} \|u(x_i)\|_2^2 \cdot vol(i)}} \tag{4.12}$$

4.1.3 Configuration du problème

On prend la gravité comme influence sur le système. Soit $g = 10000\ \mathrm{m\,s^{-2}}$ et $G = (0, -g, 0)$ le champ de gravité, $\rho = 2699\ \mathrm{kg\,m^{-3}}$ la densité, $E = 70 \cdot 10^9\ \mathrm{N\,m^{-2}}$ le module de Young, $\nu = 0.33$ le coefficient de poisson, $h = 2$ m la dimension de la poutre carrée, et $L = 8$ m sa longueur.

On trouve I d'après la formule du moment quadratique 4.3. Le champ de force w quant à lui est en $\mathrm{N\,m^{-1}}$, il convient donc d'intégrer la force sur une section de la poutre, ce qui donne :

$$w = -glh\rho \tag{4.13}$$

Comme nous utilisons un solveur dynamique, mais que nous souhaitons arriver à l'état stable pour comparer nos résultats à la solution analytique, nous vérifions la différence entre les positions courantes à deux étapes consécutives. Si cette différence est inférieure à un certain seuil absolu que nous fixons arbitrairement (ici à 10^{-9}), on arrête la simulation et on récupère les résultats de la dernière itération. Cela revient à tester si la première dérivée temporelle de la position est nulle ; en toute rigueur nous devrions utiliser la seconde, mais en pratique le seuil est fixé assez faiblement pour que cela soit suffisant.

L'utilisation de ce système de seuil signifie que le nombre d'étapes avant d'arriver à l'état final peut varier selon qu'on utilise ou non l'adaptation ; on observera cependant que le temps nécessaire pour converger est similaire dans les deux cas, sauf lorsque l'on a des oscillations d'adaptations comme on peut le voir à la figure 4.8.

L'option d'utiliser un solveur statique a été envisagée puis écartée car l'évolution du système dynamique dépend des adaptations qui sont réalisées à chaque itération. et influe sur le déroulement de la simulation. On remarque en outre des instabilités avec ce solveur et l'adaptation.

4.1.4 Résultats

On présente dans le graphe 4.3 l'erreur en ordonnée selon le nombre de DDL utilisés en abscisses. On a également le temps moyen par itération en fonction du nombre de DDL au graphe 4.4. À titre d'exemple, on présente la déflexion sur l'axe moyen pour la solution analytique, le simulateur sans adaptation et celui utilisant l'adaptation pour 208 DDL à la figure 4.5 et pour 756 DDL à la figure 4.6. On présente également le temps par itération à la figure 4.7 qui contient aussi la courbe du nombre d'adaptations (décimations et subdivisions) par itération.

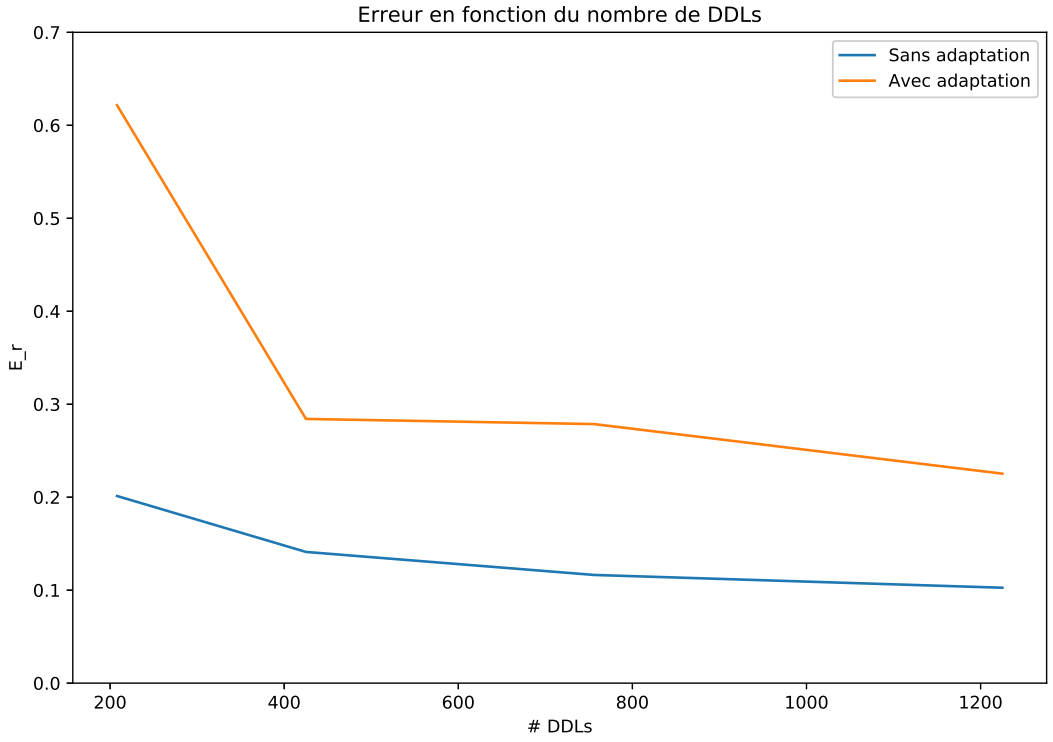

Figure 4.3 Erreur en fonction du nombre de DDL

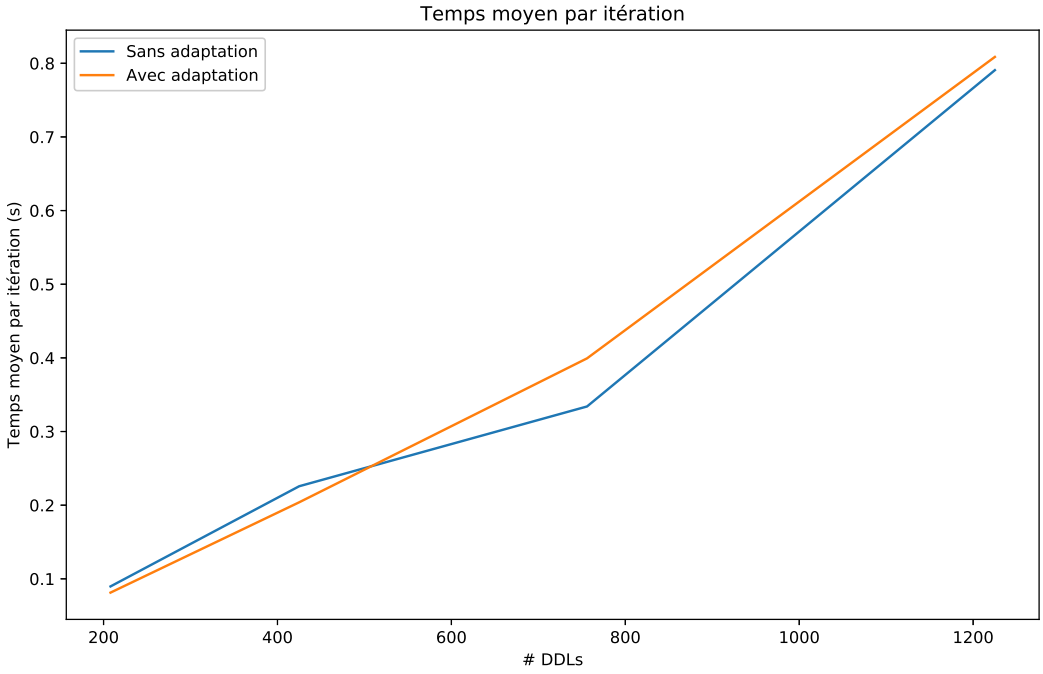

Figure 4.4 Temps moyen par itération en fonction du nombre de DDL

On présente la topologie de la poutre adaptée à 1225 DDL à la figure 4.9. Nous y avons affiché les DDL (en vert) ainsi que les estimations d'erreur qui sont les points bleus situés aux positions des PI. Les points rouges sont les DDL fixés par nos conditions aux limites. On remarque que les zones les plus subdivisées correspondent à celles dont les contraintes sont les plus importantes comme nous le confirme la figure 4.10 (tirée de [16]) et la figure 4.9. De plus, il apparaît que les estimations d'erreur sont bien uniformément réparties. On peut faire la comparaison avec la poutre non subdivisée (sans adaptation), qui possède des estimations d'erreur plus importantes en début de poutre à la figure 4.11. Cela montre que l'adaptation répartit mieux l'erreur sur l'ensemble de l'objet que le système initial.

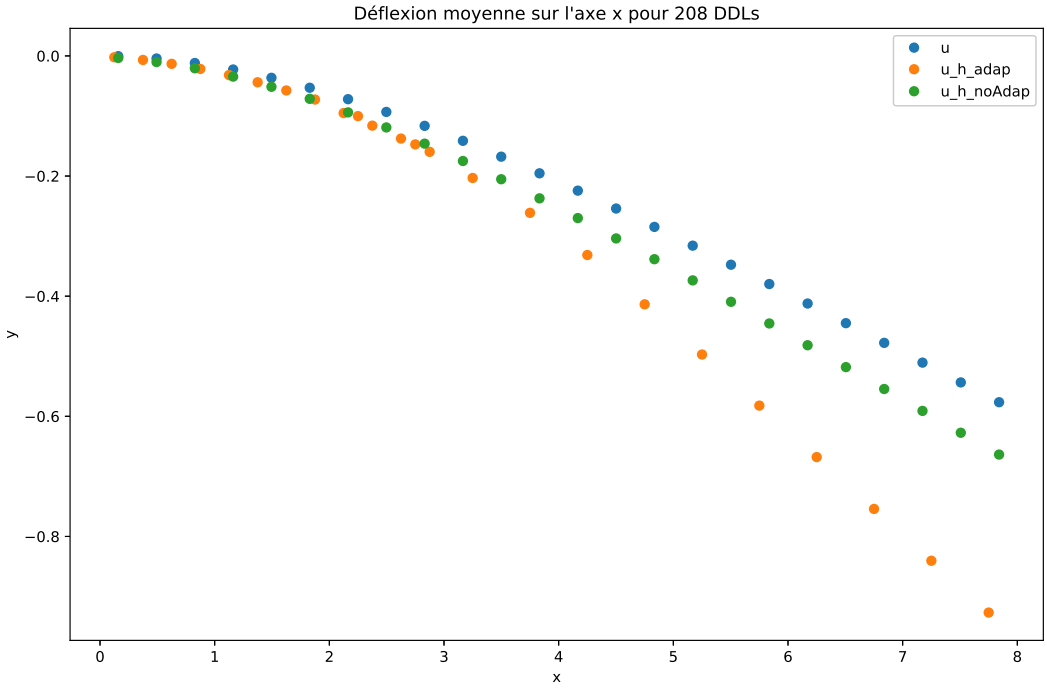

Figure 4.5 Déflexion moyenne en mètres avec et sans adaptation pour 208 DDL

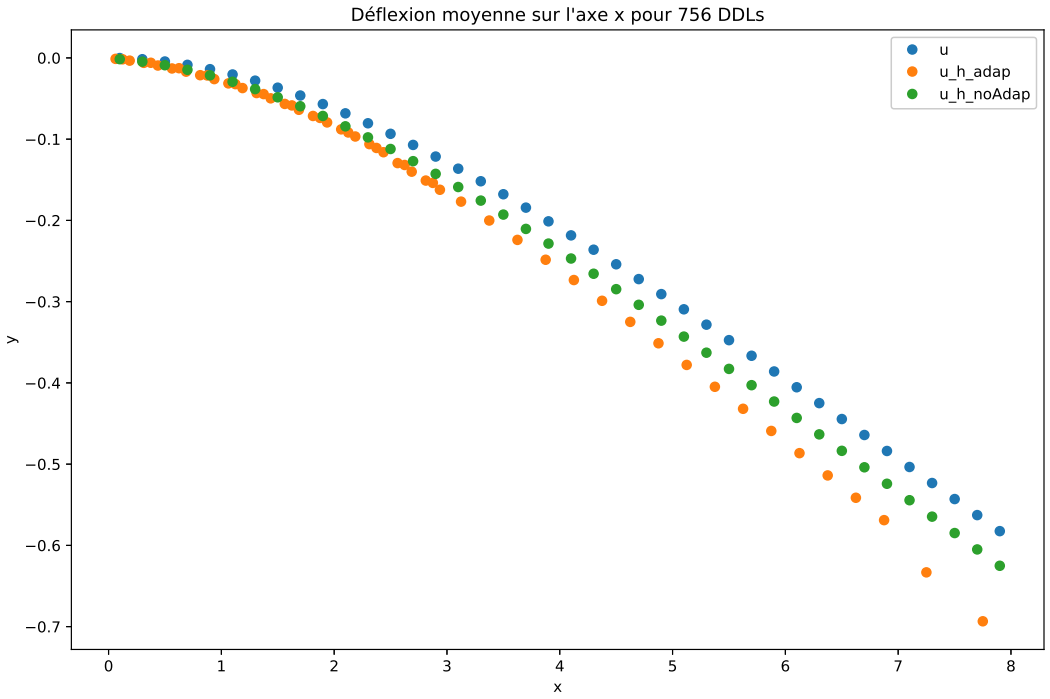

Figure 4.6 Déflexion moyenne en mètres avec et sans adaptation pour 756 DDL

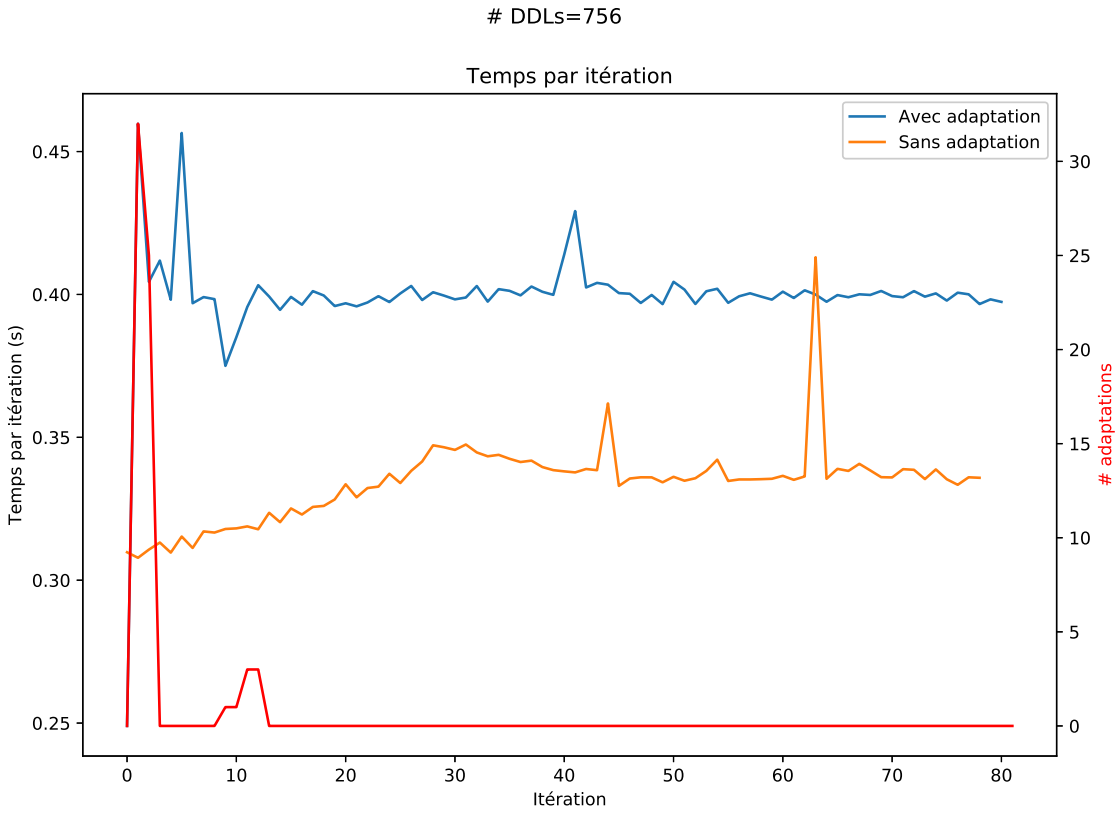

Figure 4.7 Temps par itération pour 756 DDL

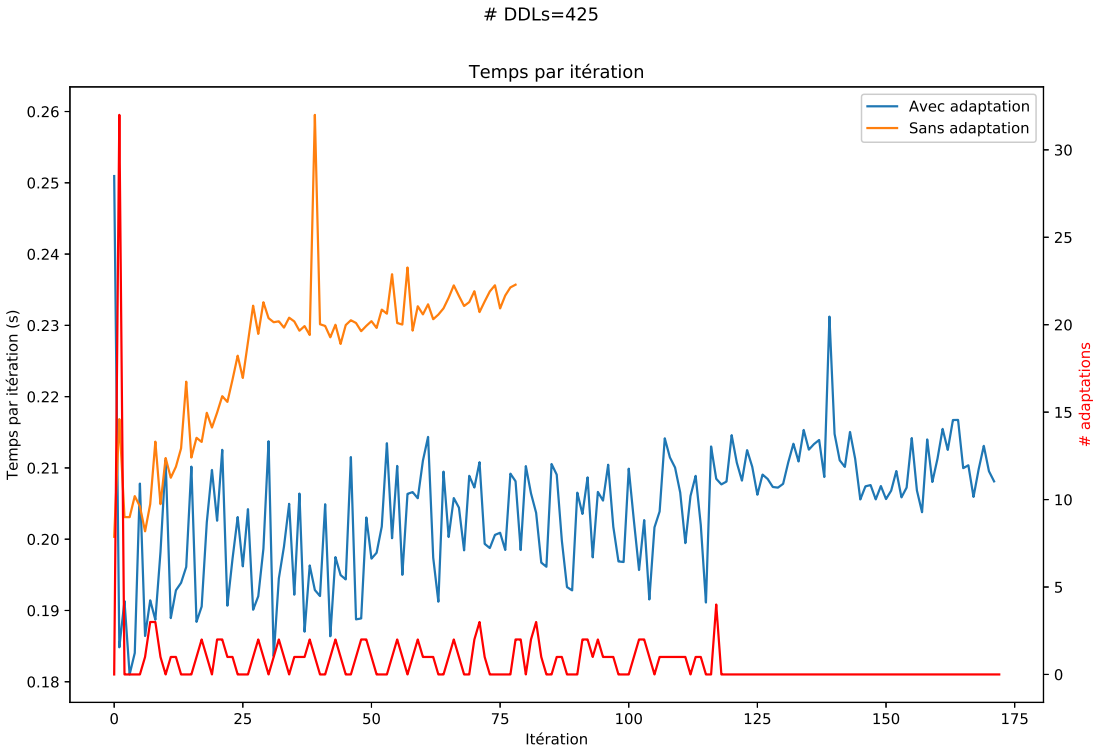

Figure 4.8 Temps par itération pour 425 DDL

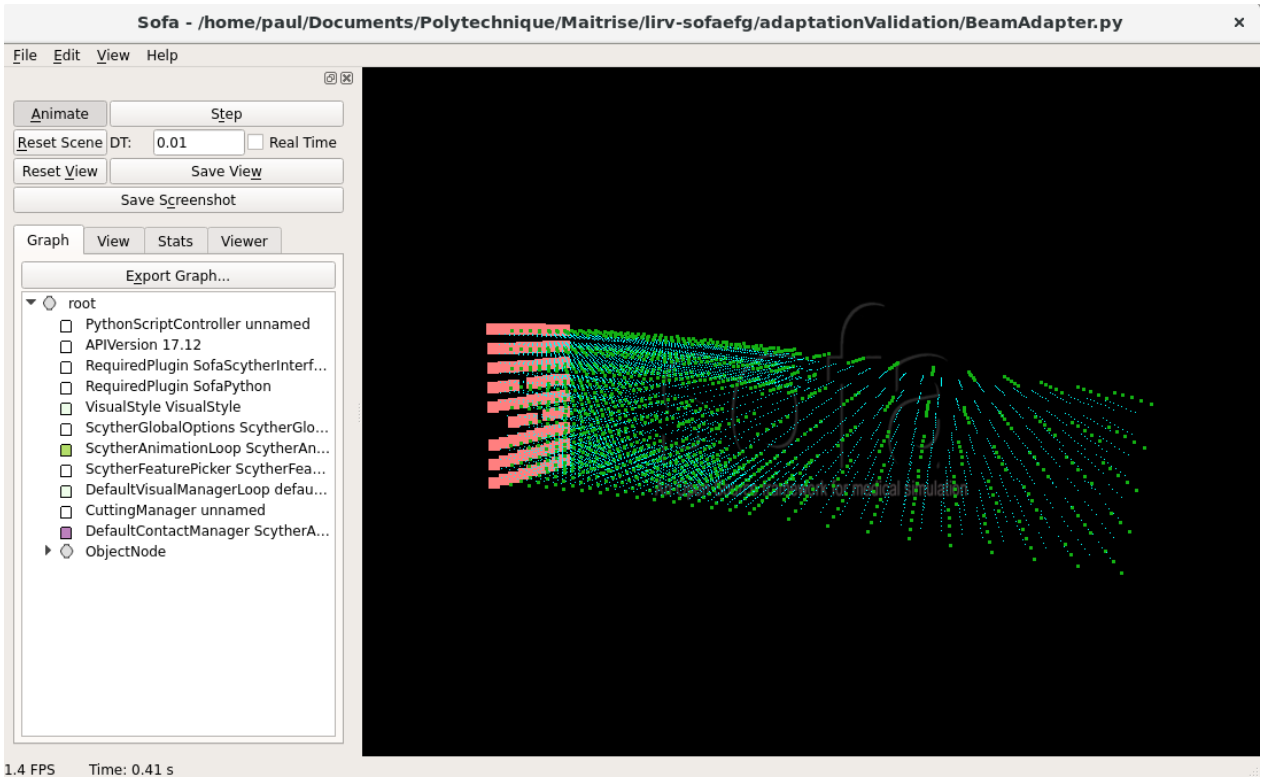

Figure 4.9 Topologie de la poutre adaptée à 1225 DDL

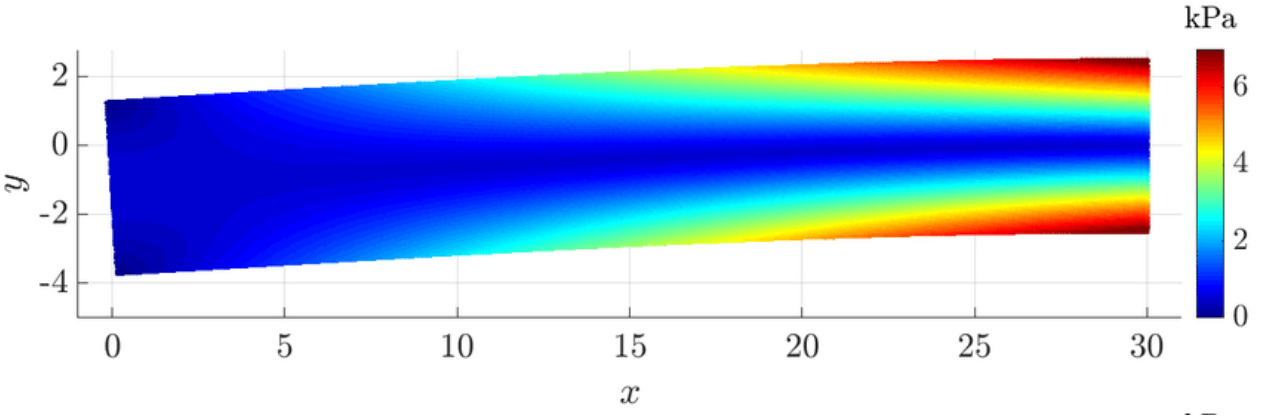

Figure 4.10 Répartition des contraintes dans une poutre encastrée

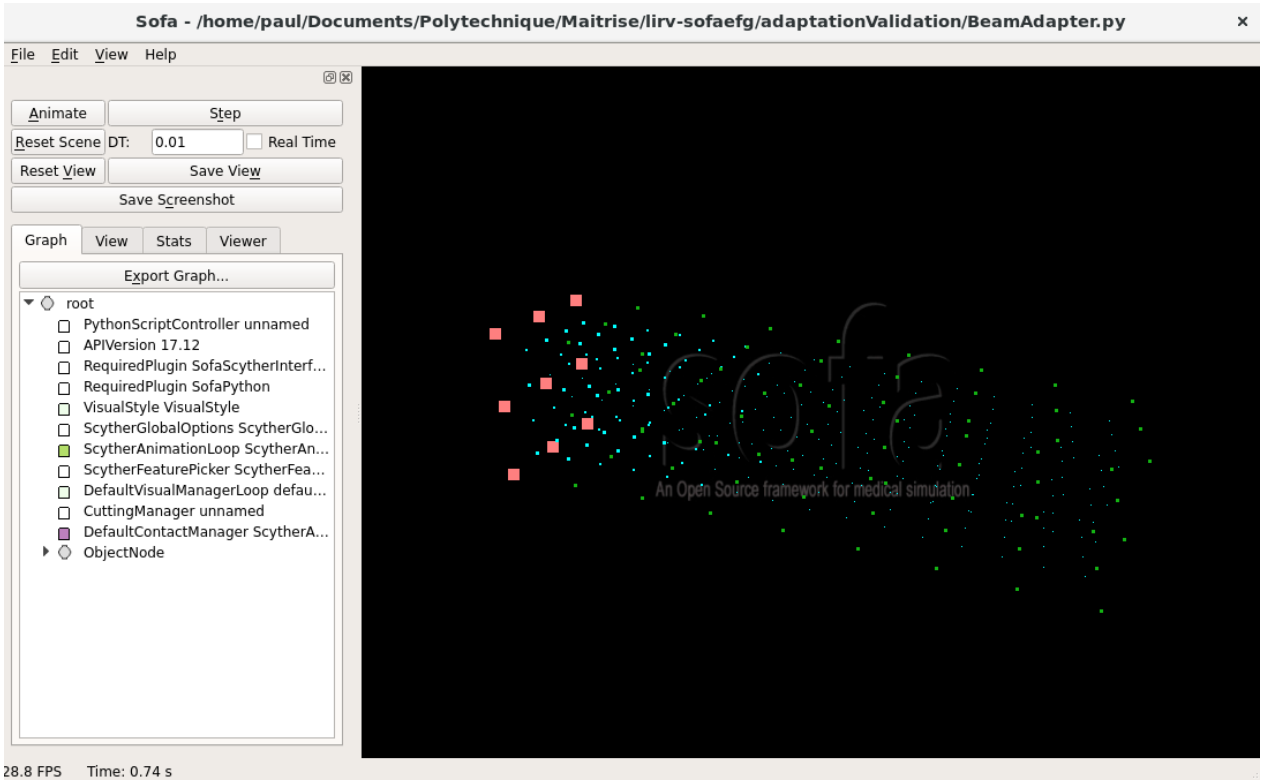

Figure 4.11 Exemple de mauvaise répartition de l'erreur

4.1.5 Discussion

L'hypothèse H2 1.2 stipulant que le temps de calcul ne devrait pas dépasser 10% du temps de calcul sans adaptation est vérifiée : on a une différence de temps de calcul de 4.2% en moyenne, avec pour certains cas une amélioration comme nous pouvons le voir à la figure 4.4. Cela semble a priori très étrange étant donné que nous avons rajouté des composantes au simulateur qui nécessitent des opérations supplémentaires, et que nous n'avons pas effectué d'optimisation du solveur. De plus, on peut même observer à la figure 4.8 que le temps par itération est presque toujours plus bas avec l'adapteur que sans. On remarque néanmoins l'inverse à la figure 4.7. Cela semble suggérer que la présence d'adaptations dans le simulateur est négligeable par rapport à d'autres variables explicatives ; un autre facteur pourrait venir du solveur en lui-même. Selon la topologie et les déformations, il peut être plus lent à converger à chaque itération pour faire l'inversion du système linéaire. Nous avons vérifié que le temps moyen était toujours similaire avec d'autres configurations du problème (en changeant le module de Young par exemple), et les observations restent proches. Nous pouvons donc conclure que H2 est vérifiée, et que de surcroît l'influence de l'adaptation est quasi négligeable par rapport à d'autres facteurs dans le simulateur.

Intéressons-nous à l'influence des adaptations sur le temps de calcul d'une itération. On observe à la figure 4.8 que les adaptations n'ont pas d'influence significative, tout comme dans la configuration de la figure 4.7.

Au niveau du fonctionnement de l'adaptation, on remarque sans surprise un grand pic d'adaptations au début des simulations pour les figures 4.7 et 4.8. On remarque néanmoins que les adaptations semblent causer des instabilités à la figure 4.8 car le temps d'arrivée à la convergence est largement augmenté par rapport au simulateur sans adaptation. Cela suggère que nous devons renforcer l'hystérésis empêchant ce genre de comportements dans l'adaptation. Ce phénomène n'a cependant pas été observé pour les autres nombres de DDL.

Contrairement à l'hypothèse H1 stipulant que l'on devrait avoir un gain de précision de 10% pour un même nombre de DDL, notre système augmente l'erreur. Nous tenterons d'expliquer cet état de fait en analysant ses causes possibles.

Tout d'abord, examinons les figures 4.5 et 4.6. On observe qu'à l'interface entre les différents niveaux d'adaptation (visibles grâce à la densité variable de points sur les figures), la pente de la poutre change brusquement. Ce phénomène est beaucoup plus marqué à la figure 4.5. Il semblerait que la liaison entre ces deux parties du nuage de points nécessite plus de liaisons pour augmenter la rigidité du système. On peut se rappeler qu'en effet nous n'avons pas

relié les PI des voisins non subdivisés sur les arêtes des octants subdivisés (comme nous l'avons déjà mentionné en 3.3.3). En outre, nous ne relions pas du tout les octants ayant deux niveaux de subdivision de différence, ce qui pourrait aussi diminuer le couplage entre des octants adjacents et diminuer la rigidité. Ce cas de figure a été envisagé par de Rabzuk et al. [13] sous la forme de la règle de Devloo et Oden que nous avons décrite dans la revue de littérature à la section 2.5.3. Nous ne l'avons pas implémentée en adaptation relative car cela nous oblige à connaître à l'avance le nombre de subdivisions/décimations nécessaires pour la vérifier, ce qui est réalisable, mais nécessite des structures de données supplémentaires. Nous l'avons en revanche implémentée dans l'adaptation par seuils.

Néanmoins, on remarque aussi que plus le nombre de DDL augmente, plus la rigidité dans la MGSM est grande (à l'inverse de la MEF). Cela suggère que les placements des nouveaux DDL sont bons car ils sont situés aux lieux subissant le plus de contraintes comme nous pouvons le voir à la figure 4.9, soit en début de poutre, qui vont par conséquent avoir une forte influence sur la déflexion maximale à son extrémité. Nous observons cependant qu'il arrive que les subdivisions ne soient pas symétriques (comme à la figure 4.9) car il manque des DDL en réserve pour cela. Tout comme pour la MEF, un maillage asymétrique augmentera l'erreur pour un problème symétrique.

Plus généralement, nous pouvons deviner qu'une des limites de ces analyses est la capacité de calcul. En effet, nous avons été limités à un maximum de 1225 DDL pour nos tests de validation. Pour contrer cela, nous avons vérifié que les résultats sont similaires pour des valeurs des variables du problème différentes. Nous présentons ces résultats dans les figures 4.12.

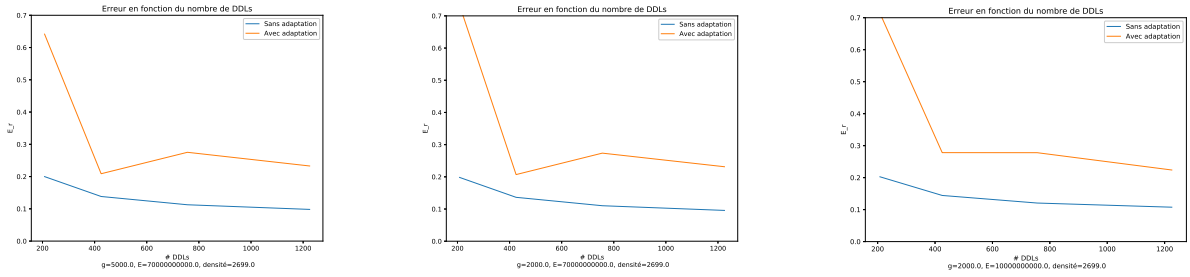

Figure 4.12 Erreur en fonction du nombre de DDL des paramètres alternatifs

4.2 Analyse qualitative

Le solide déformable présenté dans ce scénario est le cube de la figure 3.17. Nous appliquons manuellement via la souris des déformations sur des points choisis arbitrairement. La vidéo disponible en contenu additionnel à ce mémoire présente le déroulement de la simulation et une analyse temporelle est proposée dans la figure 4.13.

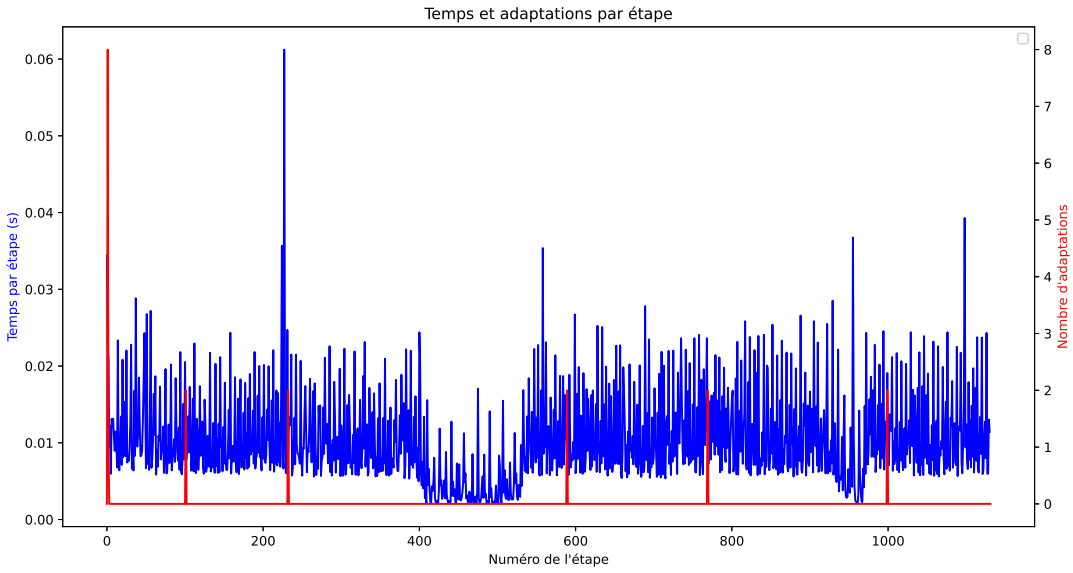

Figure 4.13 Temps et adaptations par étapes du cube contraint à des déformations manuelles

On remarque de nombreuses adaptations (subdivisions et décimations indifférenciées) à la première itération. Cela concorde avec le système d'adaptation relative qui possède à l'initialisation un grand nombre de subdivisions pour arriver au nombre maximal de DDL. Ensuite, on a des pics de 2 adaptations (une subdivision et une décimation) lorsqu'on déforme le cube : elles correspondent à des relocalisations de DDL.

Au niveau des performances par itération, il n'est pas clair que les adaptations requièrent une surcharge momentanée de temps de calcul comme cela avait déjà été vérifié lors du scénario précédent. On nuancera ce point en considérant que très peu d'adaptations sont effectuées à la fois. On remarque en outre certaines chutes de temps de calcul après une certaine durée sans perturbations (après le 3e pic et le 5e pic). Cela provient du fait que plus l'objet est stable, plus le solveur du système linéaire de SOFA converge rapidement.

4.3 Atteinte des objectifs

Rappelons les objectifs que nous nous étions fixés :

- Objectif principal : Développement d'une méthode adaptative permettant d'améliorer l'efficience de la MGSM.
- O1 : Implémentation d'un estimateur d'erreur ;
- O2 : Implémentation d'un critère de subdivision ;
- O3 : Implémentation d'un critère de décimation ;
- O4 : Subdivision du maillage de DDL et de cellules d'intégration ;
- O5 : Décimation du maillage de DDL et de cellules d'intégration ;
- O6 : Calcul des composantes des DDL et PI après subdivision ;
- O7 : Calcul des composantes des DDL et PI après décimation.
- O8 : Vérification de la méthodologie développée avec les hypothèses suivantes :
  - Hypothèse H1 : la nouvelle méthode ne nécessite pas pas plus de 10% de temps de calcul supplémentaire par rapport à l'algorithme non adaptatif pour un même nombre de DDL.
  - Hypothèse H2 : la nouvelle méthode permet de réduire l'erreur d'au moins 10% par rapport à la version non adaptative de l'algorithme pour un même nombre de DDL.

L'objectif O1 a été atteint comme nous pouvons le voir lors des deux scénarios décrits précédemment. Son expression est définie dans la formule 3.2. Les objectifs O2 et O3 ont quant à eux été déclinés en deux variantes pour l'adaptation par seuils 3.3.2 et l'adaptation relative 2.4.3. O4 est décrit dans la section 3.3.3 et O5 dans la section 3.3.4. On peut de plus les voir en action dans la section 4.2. Pour O6 et O7, on décrit les procédures aux sections 3.3.3 et 3.3.4.

On remarque cependant que l'objectif O8 n'a pas été validé, puisque notre système augmente l'erreur au lieu de la diminuer. Ainsi notre objectif principal n'est pas non plus atteint.

CHAPITRE 5 CONCLUSION

5.1 Synthèse des travaux

Nous avons présenté un système d'adaptation se greffant sur la MGSM dans le but d'en améliorer la précision tout en limitant son coût en temps de calcul. Nous avons avant toute chose fait une reformulation de la MGSM pour spécifier toutes les approximations du simulateur préexistantes à notre travail. Cela nous a permis de poser les bases théoriques pour explorer la littérature traitant de l'adaptation dans la MGSM ; nous en avons repris certaines idées. La description de la méthode d'adaptation a ensuite été divisée en plusieurs modules indépendants : l'estimation de l'erreur, le critère d'adaptation décidant des lieux où subdiviser ou décimer le nuage de DDL, ainsi que les procédures de subdivision et de décimation elles-mêmes. Deux critères d'adaptation ont été implémentés : le premier, plus simple, repose sur des seuils arbitraires d'estimation de l'erreur qui enclenchent les mécanismes d'adaptation ; le second s'affranchit de cette limitation et propose de ne demander que le nombre maximal de DDL utilisables pour les répartir de façon à ce que l'estimateur d'erreur soit le plus homogène sur l'ensemble de l'objet. Selon le critère choisi, on décide de densifier ou raréfier les DDL dans une certaine zone de l'objet. Cette étape demande d'effectuer un certain nombre de vérifications pour éviter la duplication de DDL ainsi que pour déterminer les voisinages des nouveaux éléments générés. Finalement, nous avons réalisé des tests pour deux scénarios : un premier, quantitatif, permettant de mesurer la performance en temps de calcul et en précision de l'adaptation par rapport au système initial en les comparant à une solution analytique, et un second, qualitatif, présentant des déformations manuelles suivies d'adaptations dynamiques. Ces tests nous ont permis de valider la plupart de nos objectifs spécifiques, mais l'objectif principal qui était d'améliorer l'efficience de la MGSM n'a pas été atteint.

5.2 Limitations de la solution proposée

L'hypothèse H1 1.2 du gain de précision de 10% n'a pas été validée car le module d'adaptation introduit de l'erreur dans le système. Cela constitue une limite à son utilisation car il s'agit de sa fonction première. Il est néanmoins désormais possible de modifier les sous-modules d'adaptivité pour implémenter d'autres critères ou méthodes d'adaptation, ce qui pourrait servir à changer l'objectif de notre module ou à améliorer ses performances.

5.3 Améliorations futures

Nous avons vu dans la section 4 que certaines méthodes dans l'adaptation pourraient être problématiques. On peut citer par exemple les voisinages sur les arêtes ou la règle de Devloo et Oden pour le critère d'adaptation relative. Autrement, les résultats ont été obtenus sur des formes ne possédant pas d'octants partiellement remplis. Il pourrait être intéressant de donner la possibilité au module d'adaptation de subdiviser ou décimer ces octants pour donner plus de flexibilité au système. En outre, une investigation plus approfondie des causes du manque de rigidité du simulateur doit être menée. Il se pourrait que les liaisons doivent être renforcées à l'interface entre deux niveaux d'adaptation différents.

Dans un autre registre, l'adaptation pourrait être couplée avec le module de découpe développé par Vincent Magnoux [3]. Le critère d'adaptation pourrait être modifié pour donner plus de précision à la découpe juste avant que la lame rentre en contact avec l'objet. Cela demanderait des ajustements au niveau des deux méthodes mais la connectivité entre les DDL, très importante pour la découpe, est déjà fonctionnelle avec l'adaptation.

RÉFÉRENCES

[1] M. J. Turner *et al.*, “Stiffness and Deflection Analysis of Complex Structures,” *Journal of the Aeronautical Sciences*, vol. 23, n°. 9, p. 805–823, 1956. [En ligne]. Disponible : https://doi.org/10.2514/8.3664

[2] T. Belytschko *et al.*, “Meshless methods : An overview and recent developments,” *Computer Methods in Applied Mechanics and Engineering*, vol. 139, n°. 1-4, p. 3–47, déc. 1996. [En ligne]. Disponible : https://linkinghub.elsevier.com/retrieve/pii/S004578259601078X

[3] V. Magnoux et B. Ozell, “Real-time visual and physical cutting of a meshless model deformed on a background grid,” *Computer Animation and Virtual Worlds*, vol. n/a, n°. n/a, p. e1929, 2020. [En ligne]. Disponible : https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.1929

[4] M. Kamranian, M. Dehghan et M. Tatari, “An adaptive meshless local Petrov–Galerkin method based on a posteriori error estimation for the boundary layer problems,” *Applied Numerical Mathematics*, vol. 111, p. 181–196, janv. 2017. [En ligne]. Disponible : http://www.sciencedirect.com/science/article/pii/S016892741630174X

[5] T. Belytschko, Y. Y. Lu et L. Gu, “Element-free galerkin methods,” *International Journal for Numerical Methods in Engineering*, vol. 37, n°. 2, p. 229–256, 1994. [En ligne]. Disponible : https://doi.org/10.1002/nme.1620370205

[6] J.-N. Brunet *et al.*, “Corotated meshless implicit dynamics for deformable bodies,” dans *27. International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*. Pilsen : World Society for Computer Graphics, mai 2019. [En ligne]. Disponible : https://dx.doi.org/10.24132/CSRN.2019.2901.1.11

[7] A. Fortin, *Les éléments finis : de la théorie à la pratique*. École polytechnique de Montréal, 1997-2006.

[8] Z. Ullah et C. Augarde, “Finite deformation elasto-plastic modelling using an adaptive meshless method,” *Computers & Structures*, vol. 118, p. 39–52, mars 2013. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/abs/pii/S0045794912000879

[9] L. Gavete, J. L. Cuesta et A. Ruiz, “A procedure for approximation of the error in the EFG method,” *International Journal for Numerical Methods in Engineering*, vol. 53, n°. 3, p. 677–690, janv. 2002. [En ligne]. Disponible : http://doi.wiley.com/10.1002/nme.307

[10] Y. Luo et U. Häussler-Combe, “A gradient-based adaptation procedure and its implementation in the element-free Galerkin method,” *International Journal for Numerical Methods in Engineering*, vol. 56, n°. 9, p. 1335–1354, 2003. [En ligne]. Disponible : https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.615

[11] U. Häussler-Combe et C. Korn, “An adaptive approach with the Element-Free-Galerkin method,” *Computer Methods in Applied Mechanics and Engineering*, vol. 162, n°. 1, p. 203–222, août 1998. [En ligne]. Disponible : http://www.sciencedirect.com/science/article/pii/S0045782597003447

[12] Y. Krongauz et T. Belytschko, “Consistent pseudo-derivatives in meshless methods,” *Computer Methods in Applied Mechanics and Engineering*, vol. 146, n°. 3, p. 371–386, juill. 1997. [En ligne]. Disponible : http://www.sciencedirect.com/science/article/pii/S0045782596012340

[13] T. Rabczuk et T. Belytschko, “Adaptivity for structured meshfree particle methods in 2d and 3d,” *International Journal for Numerical Methods in Engineering*, vol. 63, n°. 11, p. 1559–1582, 2005. [En ligne]. Disponible : https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1326

[14] J. Yvonnet *et al.*, “A simple error indicator for meshfree methods based on natural neighbors,” *Computers & Structures*, vol. 84, n°. 21, p. 1301–1312, août 2006. [En ligne]. Disponible : http://www.sciencedirect.com/science/article/pii/S0045794906001258

[15] G. R. Joldes, A. Wittek et K. Miller, “Adaptive numerical integration in Element-Free Galerkin methods for elliptic boundary value problems,” *Engineering Analysis with Boundary Elements*, vol. 51, p. 52–63, févr. 2015. [En ligne]. Disponible : http://www.sciencedirect.com/science/article/pii/S0955799714002495

[16] J. Slak et G. Kosec, “Parallel coordinate free implementation of local meshless method,” dans *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, mai 2018. [En ligne]. Disponible : https://doi.org/10.23919/MIPRO.2018.8400034

ANNEXE A MULTIPLICATEURS DE LAGRANGE POUR LA MGSM

Considérons le problème suivant sur le domaine Ω de frontière Γ :

$$\begin{cases} \nabla\dot{\sigma}(x) + b(x) = 0, \ x \in \Omega \\ \sigma(x) \cdot n = \bar{t}, \ x \in \Gamma_t \\ u(x) = \bar{u}(x), \ x \in \Gamma_u \end{cases}$$

Où σ est le tenseur des contraintes correspondant au déplacement u et b est le champ de vecteurs appliqué au corps.

La forme faible de ce problème est : soit $u \in H^1$, les multiplicateurs de Lagrange $\lambda \in H^0$, et les fonctions de test $\delta v \in H^1$ et $\delta\lambda \in H^0$. Si :

$$\forall \delta v \in H^1, \delta\lambda \in H^0, \int_\Omega \delta(\nabla_s v^T) : \sigma d\Omega - \int_\Omega \delta v^T \dot{b} d\Omega - \int_{\Gamma_t} \delta v^T \dot{\bar{t}} d\Gamma - \int_{\Gamma_u} \delta\lambda^T \dot{(u - \bar{u})} d\Gamma - \int_{\Gamma_u} \delta v^T \dot{\lambda} d\Gamma = 0$$

Alors les conditions d'équilibre et les conditions aux limites sont satisfaites. Ici $\nabla_s v^T$ est la partie symétrique de ∇v^T. H^1 et H^0 sont des espaces de Sobolev d'ordre un et zéro respectivement. Les multiplicateurs de Lagrange permettent d'imposer les conditions aux limites.

On obtient un système d'équations discrètes à partir de la formulation faible, on approxime u et δv par les moindres carrés mobiles dans la MGSM. En ce qui concerne les multiplicateurs de Lagrange, on les pose comme suit :

$$\lambda(x) = N_I(s)\lambda_I, x \in \Gamma_u$$

$$\delta\lambda(x) = N_I(s)\delta\lambda_I, x \in \Gamma_u$$

En utilisant la convention d'Einstein pour les indices répétés.

On peut dès lors construire le système d'équations discrètes de la même façon que pour les éléments finis.

ANNEXE B MÉTHODE MESHLESS-LOCAL-PETROV-GALERKIN (MLPG)

Une fonction $u \in H_0^1(\Omega)$ est solution faible de l'équation

$$\begin{cases} \mathcal{L}u(x) = f(x), & x \in \Omega \\ u(x) = 0, & x \in \delta\Omega \end{cases} \tag{B.1}$$

si :

$$a(u, v) = l(v), \forall v \in H_0^1(\Omega) \tag{B.2}$$

où :

$$a(u, v) = \int_\Omega \mathcal{L}uv, \quad l(v) = \int_\Omega fv$$

et $\mathcal{L}$ est une fonctionnelle linéaire.

Plutôt que de considérer l'équation globale, MLPG propose de considérer un ensemble de fonctions de test v_i chacune centrée en x_i avec le support $B_{r_i}(x_i) \subset \Omega$. Ces domaines locaux se recoupent et recouvrent Ω. On peut alors écrire un système d'équations :

$$a_{B_i}(u, v_i) = l_{B_i}(v_i), \forall i \in [[1, N]] \tag{B.3}$$

en prenant $u \in Vect(\phi_i(x))_{i=1}^N$ où les ϕ_i sont les fonctions de forme.