FrameField 2.5D

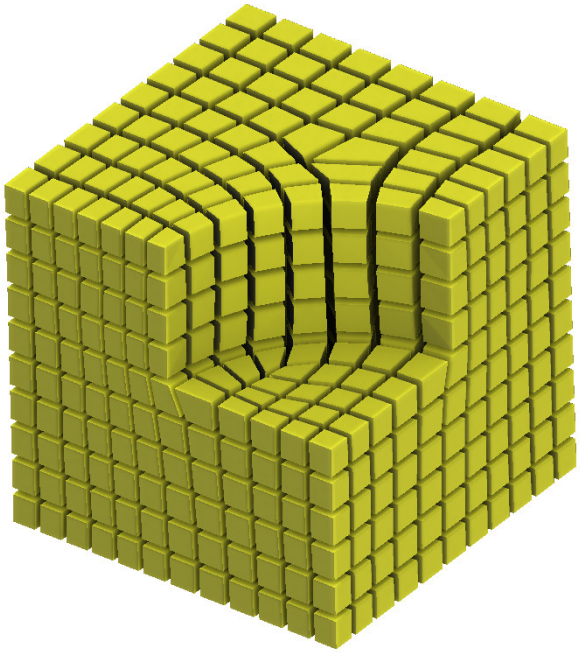
Stage de M2

Yoann Coudert–Osmont

Supervisé par

Dmitry Sokolov Nicolas Ray

Avril - Août 2020



école normale supérieure paris–saclay
MATHÉMATIQUES VISION APPRENTISSAGE

Inria

Loria
Laboratoire lorrain de recherche en informatique et ses applications

Remerciements

Quelques semaines après la signature de la convention de stage, le confinement généralisé de l'ensemble du territoire français était annoncé suite à l'épidémie de la COVID-19, conduisant à une rupture d'une grande partie de nos relations sociales et à un recentrage de nos vies sur nos familles ou nos proches. Comme pour nombre de personnes, le télétravail a été la solution adoptée pour la poursuite de ce stage. Alors que ce changement de vie parfois difficile a pu s'accompagner d'impacts psychologiques pour beaucoup, j'ai eu la chance d'être accueilli par ma famille dans la chambre de mon enfance. Mes premiers remerciements s'adressent donc à ma famille qui a su rendre cette période vivable et même agréable lors de multiples promenades pendant ces deux mois de confinement.

M'étant dirigé vers l'équipe PIXEL en grande partie pour la bonne ambiance qu'il semblait y avoir, j'ai tout de même été très agréablement surpris par leur accueil chaleureux, leur cohésion et le contact que nous avons eu durant toute cette période. Alors que de mon expérience dans la recherche ainsi que de l'expérience de plusieurs camarades, nombre de stagiaires n'ont que trop peu d'échanges avec leurs maîtres de stages, un café virtuel était organisé en visio chaque matin avec toute l'équipe. Durant le reste de la journée nous gardions le contact, principalement par messages. Dmitry s'est même débrouillé pour me faire venir à Nancy ainsi que les autre stagiaires en Juillet. J'y ai passé 10 jours très agréables et j'y ai même vécu de supers péripéties lors d'une balade à vélo. Je remercie donc grandement toute l'équipe, Dmitry, Nicolas, Étienne, Laurent, Justine et François, ainsi que les stagiaires Victor, Leïla et enfin David qui est maintenant en thèse au sein de l'équipe.

Table des matières

Introduction **4**

1 Paramétrisation globale **5**

2 Singularités et qualité d'un maillage **7**

I Génération d'un maillage hexaédrique depuis un maillage tétraédrique **10**

1 Extraction des bords **10**

2 Projection des murs sur le sol **12**
2.1 Définition d'un champ scalaire 13
2.2 Réflexion sur les contraintes aux bords 16
2.3 Projection 21

3 Déformation et projection du volume **21**

4 *framefield* 2D **23**
4.1 Création d'une base orthonormale en chaque tétraèdre 23
4.2 Génération du *framefield* 2D 25

5 Fin du *pipeline* et résultats **28**

II Correction de singularités **30**

1 Propagation des directions stables **31**

2 Mise à jour locale du *framefield* **34**

Conclusion **37**

Introduction

Aujourd'hui les simulations numériques sont omniprésentes dans l'industrie. Ces simulations nécessitent de modéliser des objets physiques au sein d'un ordinateur. Une représentation couramment utilisée est le maillage, c'est à dire la donnée d'un ensemble de points appartenant à l'objet ainsi que la donnée d'un ensemble de cellules/polytopes reliant ces points (des arrêtes, des triangles, ...). L'ensemble des cellules forme alors un pavage de l'objet.

Il existe beaucoup d'outils pour générer des maillages, mais actuellement seul la génération de maillages utilisant des simplexes (c'est à dire des polytopes à n dimensions formés par $n+1$ sommets, comme des triangles ou des tétraèdres) en tant que cellules, est relativement bien maîtrisé. Durant ce stage on s'intéresse aux maillages hexaédriques, donc, construits à partir de petits cubes déformés possédant 8 sommets et 6 faces, au lieu de 4 sommets et 4 faces pour les tétraèdres. Dans certains domaines de l'industrie les hexaèdres sont privilégiés, car ils permettent de former des maillages plus structurés et peuvent parfois donner des simulations plus performantes. Cependant la génération de tels maillages est bien plus difficile et il n'existe pas à ce jour de méthodes complètement automatiques et satisfaisantes pour le faire. Les maillages hexaédriques sont alors très souvent construits manuellement. C'est pour cela que l'équipe Inria, PIXEL s'intéresse à l'automatisation de cette tâche qui pourrait être un gagne temps conséquent. La Figure 1 permet de visualiser le côté structuré des maillages quadrangulaires et illustre une des principales difficultés rencontrées lors de la génération de ces derniers.

Il existe déjà de multiples approches pour la générations de maillages hexaédriques [Owe]. L'une d'elles, très prometteuse utilise la paramétrisation globale [1], c'est à dire le fait de créer un système de coordonnées sur une variété. Une paramétrisation globale se base usuellement sur la génération de champs de croix (*framefield*) servant de guide pour les directions des différentes coordonnées de la paramétrisation. Une branche de la croix donne la direction souhaitée d'une des coordonnées. Cela fait plusieurs années que l'équipe PIXEL explore cette méthode. Aujourd'hui cet approche est relativement bien maîtrisé en 2D [RS15], même s'il reste encore beaucoup à faire. En revanche en 3D la génération de champs de croix se complexifie. Entre autres, des champs de croix peuvent ne pas être valide en 3D. C'est à dire qu'il est possible qu'aucun maillage hexaédrique puisse correspondre à un champ de croix donné. Cela est dû au fait que l'intégration de rotations 3D est mal définie. Or certaines méthodes de génération actuelles qui se contentent de minimiser la courbure du *framefield* produisent encore des champs non valides. L'objectif de ce stage est de construire un *framefield* en 3D en extrudant une champ de croix d'une surface 2D dans une direction orthogonale (d'où le 2.5D dans le titre). Cette façon de faire permettrait notamment de garantir la validité du champ de croix.

Au cours du stage nos objectifs ont évolué. Au lieu de générer entièrement un champ de croix depuis zéro, nous avons fini par améliorer des champs générés en amont, via de précédentes méthodes, en "soignant" les singularités (définis en section 2), en les rendant valides le cas échéant. Au final ce stage aura permit de mieux comprendre ce qu'il semble possible de faire avec l'idée initiale de la 2.5D.

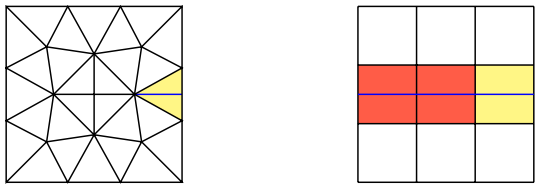


FIGURE 1 – Maillages triangulaire (à gauche) et quadrangulaire (à droite) d'un carré. Si l'on souhaite diviser en deux les cellules jaunes, on se rend compte que la forte structure du maillage quadrangulaire force à diviser en deux toute une ligne de cellules (en rouge). Il est ainsi difficile de modifier localement un maillage quadrangulaire.

1 Paramétrisation globale

Comme expliqué en introduction, la méthode utilisée dans ce stage pour construire un maillage hexaédrique est la paramétrisation globale. Cette section donne les grandes lignes de ce processus. Dans cette méthode, on cherche à construire une fonction continue et injective, f envoyant une variété $\mathbb{M}$ vers un domaine Ω d'un espace euclidien. Si on peut construire une grille s'alignant avec les bords de Ω, alors appliquer la transformation f^{-1} à cette grille permet de mailler la variété initiale $\mathbb{M}$ avec des quadrangles en 2D (illustré FIGURE 2) ou des hexaèdres en 3D.

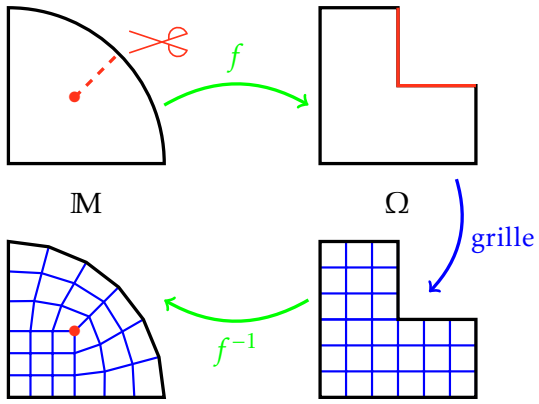


FIGURE 2 – Génération de maillages via paramétrisation globale.

On peut représenter notre fonction f par d champs scalaires lorsque l'on travaille en dimension d. Par exemple en dimension 2, on peut prendre $f = (u, v)$. Si on possède au préalable un maillage M de la variété $\mathbb{M}$ dont les cellules sont des simplexes (triangles ou tétraèdres), alors on peut définir des champs u et v par leur valeurs sur chaque sommet ou sur chaque cellule. Les lignes de nos grilles sont ensuite définies par des iso-u et des iso-v généralement prisent à valeurs entières. On remarque que pour que la grille s'aligne correctement avec les bords de $\mathbb{M}$ il faut que le gradient d'une des coordonnées soit orthogonal au bord (de normal $\vec{n}$) :

$$\forall p \in \partial\mathbb{M}, \exists \lambda \in \mathbb{R}^*, \quad \nabla u(p) = \lambda \vec{n} \text{ ou } \nabla v(p) = \lambda \vec{n} \tag{1}$$

Comme ces contraintes portent sur les gradients des champs, une idée naturelle pour construire u et v est d'effectuer une optimisation sur leurs gradients. En plus des contraintes ci-dessus on

essaie d'obtenir des champs de vecteurs les plus lisses possible pour représenter les gradients. Un critère de qualité pour les maillages est le fait que les cellules soit le plus proche possible de polygones réguliers. En 2D on veut alors que nos quadrangles ressemblent le plus possible à des carrés. Une idée pour atteindre cet objectif est d'imposer l'orthogonalité de ∇u et ∇v et de s'assurer que leur normes soient les mêmes. On cherche donc pour chaque cellule de M, deux vecteurs $\vec{z}_1$ et $\vec{z}_2$ orthogonaux indiquant les directions des gradients de u et v. Comme il est difficile de savoir pour chaque bord, s'i l sera défini par un iso-u ou un iso-v et si le gradient du champ correspondant est dirigé vers l'intérieur du modèle ou vers l'extérieur on utilisera des croix formées de quatre vecteurs $\{\vec{z}_1, \vec{z}_2, -\vec{z}_1, -\vec{z}_2\}$ invariantes par rotations de $\pi/2$. C'est ce champ de croix que l'on appelle *framefield*. Lors de l'optimisation on ne sait alors pas lequel de ces quatre vecteurs représente ∇u ni ∇v. En 2D une croix peut être représentée via un seul vecteur unitaire (ou simplement un angle), les autres s'obtenant par rotation de $\frac{\pi}{2}$ successives. En revanche en 3D, représenter une croix est plus compliqué ([RS15] propose d'utiliser une décomposition de la croix dans la base des harmoniques sphériques d'ordre 4). Le processus d'optimisation devient alors difficile et sensible aux conditions initiales.

Une fois le champ de croix obtenu, on choisi deux vecteurs orthogonaux $\vec{z}_1$ et $\vec{z}_2$ dans chaque croix afin de constituer des approximation des deux gradients ∇u et ∇v de manière cohérente. Puis vient un processus d'intégration de ces champs vectoriels pour obtenir u et v [KNP07]. L'énergie minimisée est la suivante :

$$(u, v) = \underset{\forall p \in \partial\mathbb{M},\, u(p)\in\mathbb{Z} \text{ ou } v(p)\in\mathbb{Z}}{\arg\min} \left\| (\nabla u, \nabla v) - (\vec{z}_1, \vec{z}_2) \right\|^2$$

La grille du maillage quadrangulaire est alors définie par l'ensemble des points possédant au moins une coordonnée entière. En 3D, cette étape est qualifiée de *Cube Covering* en référence à un article de 2011 [NRP11]. Finalement, les différentes étapes sont résumés dans la Figure 3.

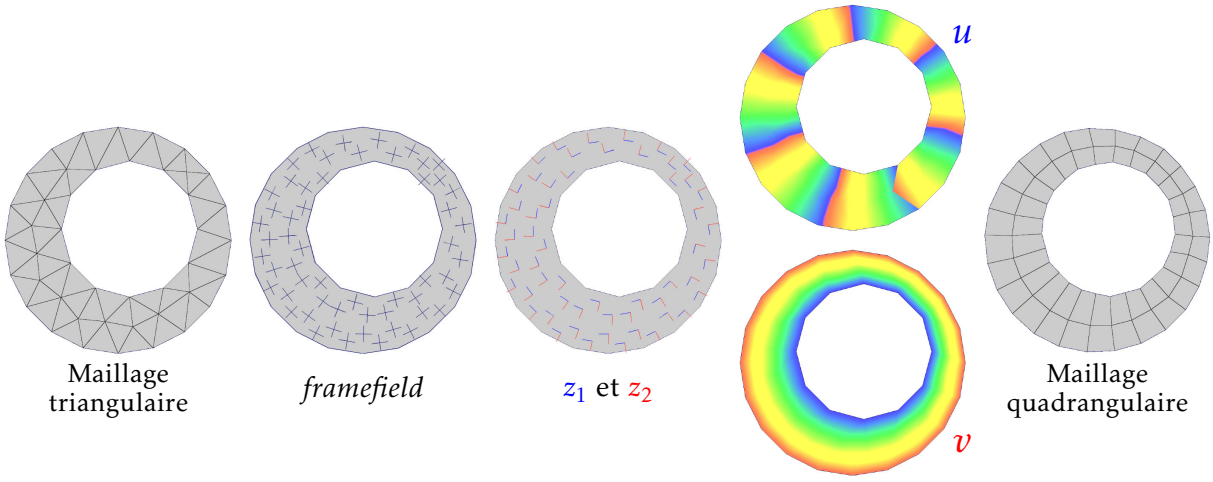


Figure 3 – Les différentes étapes de la paramétrisation globale en utilisant un *framefield*.

2 Singularités et qualité d'un maillage

Le sommet rouge de la Figure 2 me permet d'introduire la notion de singularité, très importante pour la suite de ce rapport. Dans une grille chaque sommet interne est le coin de 4 carrés. On dit que ces sommets sont de valence 4. Lorsque l'on construit un maillage quadrangulaire cette propriété n'est pas toujours possible. Il peut arriver que l'on soit contraint d'avoir des sommets de valence 3 (cas de la Figure 2) ou bien de valence strictement supérieur à 4. Ces sommets sont appelés singularités. On peut aussi définir l'indice d'un sommet i par :

$$\text{indice}(i) = \text{valence}(i) - 4$$

Si toute les singularités sont internes au maillage alors la somme de leurs indices est entièrement déterminée par la caractéristique d'Euler $\chi(M)$ et par la donnée des angles entre toutes les arrêtes du bord ∂M via la formule [FSM+17] :

$$\sum_{i \in M} \text{indice}(i) = -4\chi(M) + \sum_{i \in \partial M} (2 - n_i) \qquad \text{avec } \; n_i = \underset{0 \leqslant n \leqslant 4}{\arg\min} \left| \theta_i - n\frac{\pi}{2} \right| \tag{2}$$

Où θ_i est l'angle interne absolu du bord au sommet i (par exemple, pour un hexagone régulier cet angle est $2\pi/3$). On rappelle aussi que la caractéristique d'Euler est donné par :

$$\chi = 2 - 2g - |\mathcal{C}(\partial M)|$$

Où g est le genre de la surface et $|\mathcal{C}(\partial M)|$ est le nombre de composantes connexes du bord de M. Cette formule nous montre que la quadrangulation de certaines surfaces doit nécessairement avoir des singularités (toujours dans l'hypothèse qu'on ne place pas de singularités sur les bords, ce qui est le cas pour les polycubes par exemple [GSZ11]).

Exemple de la Figure 2

$\chi = 2 - 2 \cdot 0 - 1 = 1$

$\sum_{i \in \partial M} (2 - n_i) = 3 \cdot (2 - 1) = 3$

$\Rightarrow \sum_{i \in M} \text{indice}(i) = -4 \cdot 1 + 3 = -1$

Exemple de la Figure 3

$\chi = 2 - 2 \cdot 0 - 2 = 0$

$\sum_{i \in \partial M} (2 - n_i) = 0$

$\Rightarrow \sum_{i \in M} \text{indice}(i) = -4 \cdot 0 + 0 = 0$

En 3D, les singularités ne sont plus des sommets mais des arrêtes et la valence est alors définie comme le nombre d'hexaèdres adjacents. Une arrête régulière possède une valence de 4. Les arrêtes qui ne vérifient pas cette propriété sont singulières (exemple Figure 4 avec une valence 5). Dans un maillage hexaédrique, l'ensemble des arrêtes singulières forme un graphe appelé graphe de singularités. Une propriété notable de ce graphe est qu'aucune branche ne se termine à l'intérieur du maillage. Une singularité peut donc soit être fermée comme à l'intérieur d'un tore (Figure 5), soit avoir ses deux extrémités sur les bords du maillage (Figure 6).

Dans l'optique d'une simulation numérique, il est préférable d'avoir le moins de singularités possible dans le maillage. Toutefois ce n'est pas suffisant pour obtenir un maillage de bonne qualité. En effet au delà de la topologie, la géométrie des quadrangles/hexaèdres compte. On cherche à obtenir des cellules dont le Jacobien est le plus grand possible (le Jacobien étant maximal pour un cube, nul pour un hexaèdre aplati et négatif pour un hexaèdre retourné). Les maillages 3D obtenus avec une approche polycubes ne possèdent pas de singularités mais

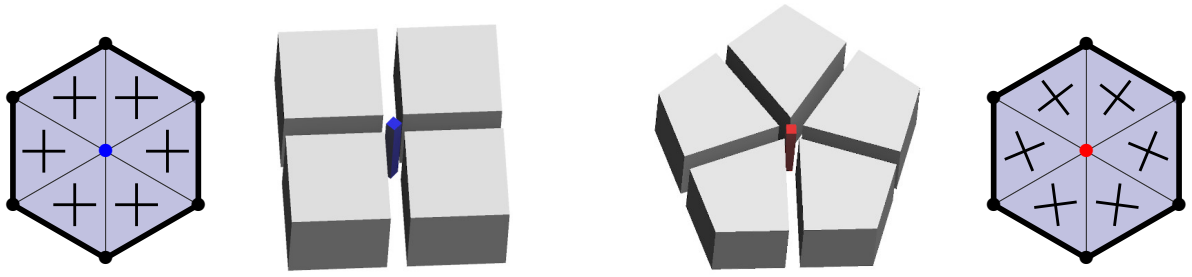


FIGURE 4 – Illustration d'une arrête régulière (à gauche) et d'une arrête singulière (à droite). Une coupe 2D du *framefield* ainsi que les hexaèdres adjacents des arrêtes en question sont donnés. Pour la singularité de valence 5, la croix subie une rotation d'un quart de tour dans le sens horaire le long d'un lacet orienté dans le sens anti-horaire.

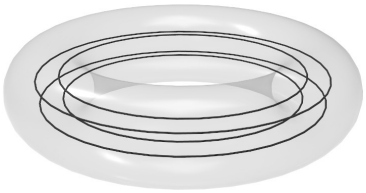


FIGURE 5 – Les quatre singularités d'un tore sont fermées.

contiennent des hexaèdres de mauvaise qualité [GSZ11]. Bien que *Cube Cover* puisse générer de mauvais hexaèdres si le graphe de singularités du *framefield* en entrée n'est pas valide, il est possible d'effectuer un prétraitement afin de restreindre le type des singularités ce qui permet de grandement augmenter le Jacobien du pire hexaèdre [LLX+12]. C'est entre autre pour la qualité de la géométrie des hexaèdres que l'approche via paramétrisation globale est intéressante. Mais comme expliqué dans le paragraphe suivant, cette approche possède un principal point bloquant.

J'ai pu évoquer la validité d'un *framefield* dans le paragraphe précédent. Ce paragraphe va expliciter cela. Dans un premier temps on peut définir de manière assez similaire les arrêtes singulières pour un *framefield* (3D) sur un maillage simplicial. Considérons une arrête e entourée de tétraèdres $(t_0, t_1, ..., t_k)$ ordonnées de sorte que t_i et t_{i+1} partagent une face. Le *framefield* associe à chacun de ces tétraèdres une croix qui contient 6 vecteurs $\{\vec{z}_1, \vec{z}_2, \vec{z}_3, -\vec{z}_1, -\vec{z}_2, -\vec{z}_3\}$ où les z_i sont unitaires et orthogonaux deux à deux. Pour chacune ce des croix on prend arbitrairement trois vecteurs u, v et w orientés dans le sens direct (i.e. $(u \times v).w > 0$). On construit ensuite la matrice orthogonale $M = \begin{pmatrix} u & v & w \end{pmatrix}$. Soit $\mathcal{G}$ le groupe des symétries (directes) du cube, on définie la meilleure permutation entre deux matrices orthogonales de même déterminant par :

$$\Pi(M, M') = \underset{P \in \mathcal{G}}{\arg\min} \left\| PM - M' \right\|_2$$

Cela nous permet de définir le type d'une arrête comme :

$$\text{type}(e) = \Pi(M_{t_0}, M_{t_1}) \circ \Pi(M_{t_1}, M_{t_2}) \circ \cdots \circ \Pi(M_{t_{k-1}}, M_{t_k}) \circ \Pi(M_{t_k}, M_{t_0})$$

Ce type représente la rotation du *framefield* le long d'un lacet autour de l'arrête e. Lorsque type$(e) = I_d$, il n'y a pas de rotation et l'arrête e est dite régulière. Si le type n'est pas l'identité l'arrête est singulière. Après passage de *Cube Cover* les singularités du *framefield* deviendront des singularités du maillage hexaédrique. Toutefois *Cube Cover* ne gère pas tous les types de singularité. Sur les 24 permutations de $\mathcal{G}$ seules 10 d'entre elles sont valides pour former un maillage hexaédrique. Les types valides sont les permutations qui laissent une direction stable. C'est à dire l'ensemble :

$$\text{Valide} = \left\{I_d, R_x, R_x^2, R_x^3, R_y, R_y^2, R_y^3, R_z, R_z^2, R_z^3\right\}$$

Où R_x est la matrice de rotation d'un quart de tour autour de l'axe x :

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}, \quad R_y = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix}, \quad R_z = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Cela vient du fait que la direction d'une arrête singulière ne peut subir de rotation le long d'un lacet autour de celle-ci. La direction de l'arrête doit ainsi rester stable le long d'un tel lacet. Un champ de croix valide en 3D est alors localement un champ de croix 2D extrudé dans une troisième direction de sorte que les singularités suivent la direction stable. Ceci est un prérequis pour que *Cube Cover* fonctionne correctement. Un *framefield* 2.5D vérifierait automatiquement cette contrainte. C'est donc sur ce point que résulte tout l'intérêt de ce stage. La Figure 6 illustre ce que l'on veut et ce que l'on ne veut pas au sein du graphe de singularités sur un modèle appelé *notch* souvent utilisé comme exemple. Les méthodes de génération de *framefield* couramment utilisées [RS15] essayent simplement d'avoir un champ le plus lisse possible. Dans cet exemple pour avoir un champ le plus lisse possible il est préférable de réduire la longueur totale des singularités en en traçant une seule reliant deux sommets contraints à être singulier (un sur la face possédant 5 coins anguleux (A) et un sur la face possédant 3 coins anguleux (B)), ce qui résulte en un graphe de singularités non valide (image de droite). Avec ce stage on espère pouvoir obtenir le graphe de singularité valide de l'image du milieu.

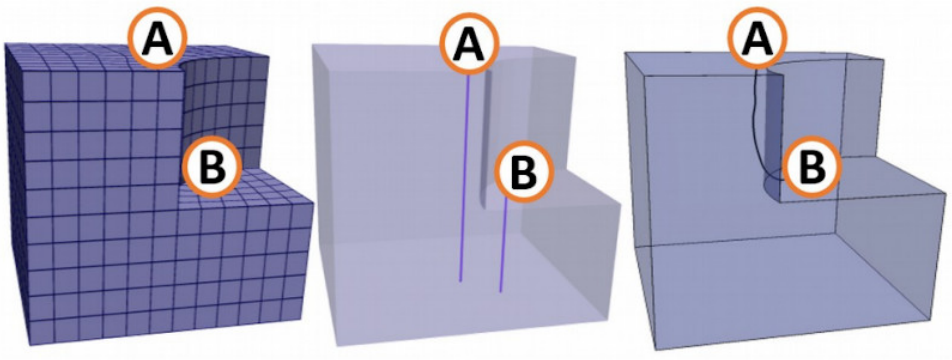


Figure 6 – Le maillage *notch* à gauche ainsi qu'un graphe de singularité valide sur ce maillage au centre et un graphe invalide à droite.

Première partie

Génération d'un maillage hexaédrique depuis un maillage tétraédrique

Dans cette partie, on souhaite construire un algorithme capable de transformer un maillage tétraédrique en un maillage hexaédrique, comme illustré Figure 7. Les entrés pouvant avoir jusqu'à quelques centaines de milliers de sommets, on va chercher à obtenir un algorithme en complexité quasi linéaire en ce nombre, en évitant des complexités quadratiques. Nous avons d'abord voulu construire un algorithme réalisant cette tâche en plusieurs étapes sans chercher directement à les optimiser, avec l'intention d'améliorer les étapes qui posent le plus de problèmes par la suite. J'utiliserai le terme anglais *pipeline* pour parler de cette succession d'étapes. Les sous-sections qui suivent présentent chacune une étape de ce *pipeline* qui peut se résumer comme suit :

- Labellisation de la surface de bord du maillage en sol/toit/mur.
- Projection des murs sur les bordures du sol.
- Extension à l'intérieur du maillage de cette projection sur le sol.
- Calcul d'un champ de croix 2D sur le sol contraint par l'ensemble du volume.
- Obtention du champ de croix 3D à l'intérieur du maillage via la projection.
- Intégration du champ de croix via *Cube Cover* et extraction des hexaèdres avec *Hexex*.

On voit apparaître l'idée d'une projection dans ce *pipeline*. En effet, l'extrusion d'un *framefield* 2D d'une surface pour remplir le volume peut aussi se voir comme la projection du volume sur une surface sur laquelle est calculé un champ de croix 2D.

1 Extraction des bords

Comme expliqué ci-dessus, on souhaite projeter le volume du maillage M sur une surface afin de travailler en 2D par la suite. Nous avons naturellement envisagé de choisir une partie du bord du maillage en tant que surface de projection. Pour cela, on divise le bord du maillage en trois catégories : le sol (∂M_s), le toit (∂M_t) et les murs (∂M_m). Tous les sommets du maillage seront alors projetés sur le sol ∂M_s. Le bord d'un maillage tétraédrique est une surface triangulé.

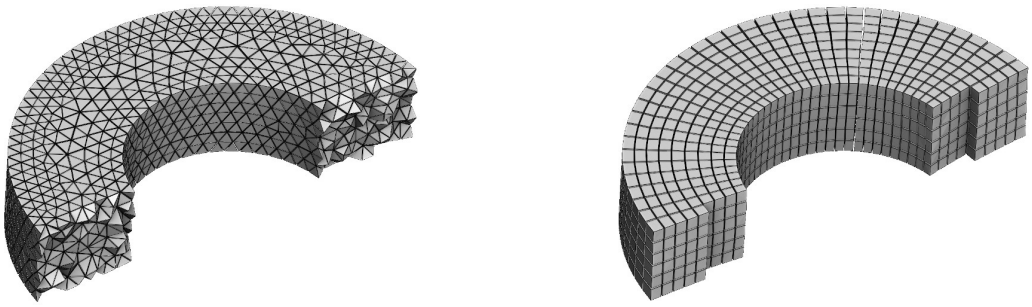


Figure 7 – Coupe d'un maillage à cellules tétraédriques à gauche, et sa transformation en cellules hexaédriques à droite.

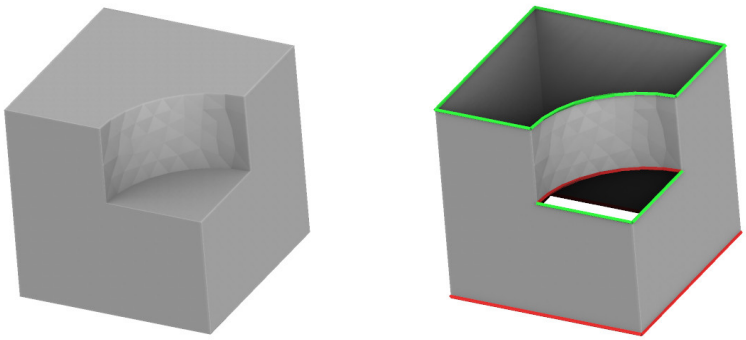


FIGURE 8 – Maillage tétraédrique à gauche et le résultat de l'extraction des murs à droite. Les arrêtes vertes correspondent au toit et les arrêtes rouges au sol.

Il nous faut donc assigner à chaque triangle de cette surface, la catégorie à laquelle il appartient, à savoir, un sol, un toit ou un mur. Avec le temps, j'ai fini par fusionner les catégories sol et toit, en la catégorie horizontale, et je marque les arrêtes qui font la jonction entre un triangle de mur et un triangle horizontal, comme étant soit une arrête de sol soit une arrête de toit. En effet certains triangles horizontaux pourront avoir à la fois des arrêtes de sol et des arrêtes de toit (voir FIGURE 8, où on trouve deux jonctions entre des arrêtes de sol et de toit dans les coins du quart de cylindre). Ainsi l'appartenance d'un triangle au toit ou au sol est assez mal défini. De plus pour simplifier le calcul de la projection, nous projetons d'abord les murs sur les arrêtes de sol et seulement ensuite nous interpolons ces projections sur l'ensemble des sommets internes du maillage, d'où l'intérêt de ne labelliser que les arrêtes décrites précédemment pour le moment.

Nous avons voulu commencer par un étiquetage très simple des faces, quitte à l'améliorer par la suite. Pour se faire on commence par choisir un vecteur vertical $\vec{z}_0$ via une heuristique ou bien manuellement. Ensuite on décide pour chaque triangle du bord s'il appartient à un mur ou s'il est horizontal en fonction de la valeur absolue du produit scalaire entre $\vec{z}_0$ et sa normale $\vec{n}$. Ainsi un triangle est horizontal si et seulement si :

$$\left| \vec{z}_0 . \vec{n} \right| > 0.95$$

S'en suit un petit traitement pour supprimer du bruît dans l'étiquetage. A la suite de ce traitement on oublie les triangles horizontaux et on ne travaille plus que sur les murs. Enfin pour chaque arrête des bords des murs, on détermine si elle appartient au sol en fonction du produit scalaire entre le vecteur vertical $\vec{z}_0$ et la hauteur $\vec{h}$ du triangle auquel appartient l'arrête (en prenant pour base cette arrête). L'appartenance au sol est alors déterminée par l'inégalité :

$$\vec{z}_0 . \vec{h} > 0$$

Un exemple d'extraction est donné, FIGURE 8.

Limitations Cet étiquetage mur/sol/toit est très simpliste. On peut imaginer des modèles qui nécessitent de séparer le maillage en plusieurs clusters ayant chacun une direction verticale différente (FIGURE 9, exemple de gauche), ou encore un modèle dans lequel la direction verticale

$\vec{z}_0$ n'est pas constante (exemple de droite). Comme il sera expliqué plus tard dans ce rapport, en cherchant à définir un meilleur étiquetage des bords du maillage, nous nous sommes rendu compte que ce que l'on essayait de faire revenait quasiment à construire un premier *framefield* dans le modèle. C'est pour cela que dans la seconde partie du rapport nous essayons d'améliorer un *framefield* au lieu d'en construire un. Ayant changé d'objectif par la suite, nous n'avons pas proposé de meilleur étiquetage que celui présenté ci-dessus.

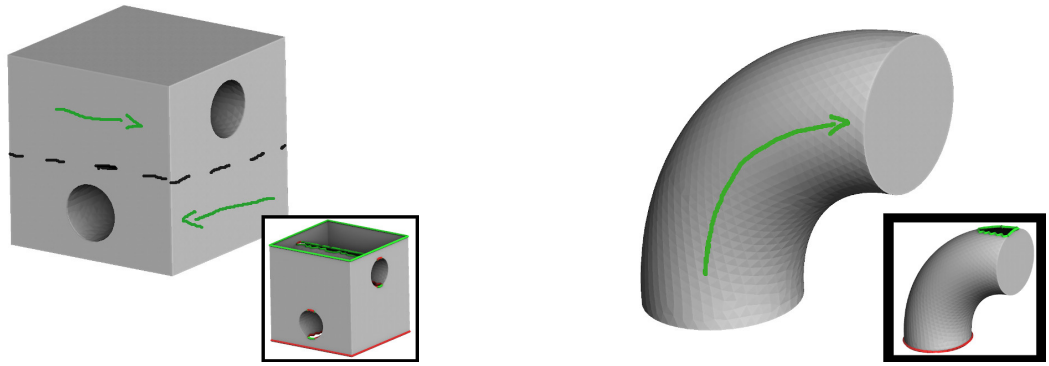


FIGURE 9 – Limitation de l'étiquetage des murs. Les flèches vertes indiquent les directions verticales que l'on aurait bien aimé avoir, et les petites vignettes montrent les résultats obtenus avec la méthode proposée.

2 Projection des murs sur le sol

Désormais nous savons sur quelles arrêtes se projetteront les murs (celles labellisées en tant que sol, en rouge sur les FIGURES 8 et 9). Une façon naturelle de faire cette projection est de définir un champ vectoriel sur les murs en direction des sols. Pour projeter un sommet, il suffit alors de suivre ce champ de vecteur jusqu'à atterrir sur une arrête de sol.

Une première idée qui m'est venue aurait été d'utiliser le champ de vecteur défini par l'opposé du gradient de la distance géodésique au sol. Ainsi un sommet du mur aurait été projeté au point du sol le plus proche. L'existence de méthodes efficaces pour calculer les distances géodésiques donne un grand avantage à ce champ. Il est par exemple possible d'utiliser l'équation de la chaleur [CWW17]. Mais il aurait fallu s'adapter à plusieurs problèmes comme le fait qu'on ne veut pas projeter des sommets sur un sol se trouvant au dessus (selon la direction verticale $\vec{z}_0$ choisie). Si on prend l'exemple du modèle *notch* de la FIGURE 8 on observe la présence d'un sol "intermédiaire" en forme de quart de cercle. Des sommets situés juste en dessous de ce sol seraient plus proche de celui-ci que du sol tout en bas du cube. On aurait alors des sommets projetés sur des arrêtes situées au dessus selon le sens de la verticalité, ce qui n'est pas souhaitable. Tous les sommets doivent être projetés dans le même sens, à savoir, vers le sol. Dans un second temps, on aimerait aussi que le champ de vecteur ait une divergence nulle. Ainsi plutôt que se baser sur une distance géodésique, on pourrait essayer d'optimiser un champ pour obtenir une divergence faible. L'équipe possède sa propre librairie de solveurs linéaires *OpenNL* développée entre autres par Bruno Lévy. C'est cette librairie qui sera utilisée pour toutes les minimisations quadratiques présentées dans ce rapport.

2.1 Définition d'un champ scalaire

Théorie dans un monde continue Nous avons une 2-variété $\partial\mathbb{M}_m$ correspondant aux murs et que l'on note ici $\mathbb{S}$. On note ensuite $\partial\mathbb{S}_s$ son bord labellisé en tant que sol et $\partial\mathbb{S}_t$ son bord labellisé en tant que toit. On veut définir un champ de vecteurs $\vec{v}$ sur la variété $\mathbb{S}$, tel que :

- La divergence du champ est faible afin d'obtenir une projection sans distorsions. On minimise alors $\int_{\mathbb{S}}(\nabla\cdot\vec{v})^2$.
- Le flux sortant à travers chaque composante connexe de sol est constant et positif. Le flux sortant de chaque composante connexe de toit est constant et négatif :
  $\forall C\in\mathcal{C}(\partial\mathbb{S}_s)\cup\mathcal{C}(\partial\mathbb{S}_t),\ (\vec{v}.\vec{n})_{|C}=c^{te}$ Où $\mathcal{C}(X)$ est l'ensemble des composantes connexes de X et $\vec{n}$ est le vecteur normal sortant au bord de $\mathbb{S}$.

Le second point permet de ne pas avoir une infinité de solution (dont la solution nulle). Ce choix de contraintes est totalement questionnable, mais ce ne fut qu'un premier essai ne semblant pas si mauvais. Ensuite, définir un champ de vecteur via un programme d'optimisation n'est pas une tâche facile. Mais cette difficulté peut être soulevée en prenant $\vec{v}$ comme le gradient d'un champ scalaire u ($\vec{v}=\nabla u$). Le problème devient alors :

$$\min_u \int_{\mathbb{S}}(\nabla\cdot\nabla u)^2=\int_{\mathbb{S}}(\Delta u)^2 \qquad \text{t.q.}\ \begin{cases}(\nabla u.\vec{n})_{|\partial\mathbb{S}_s}=1\\ \forall C\in\mathcal{C}(\partial\mathbb{S}_t),\ (\nabla u.\vec{n})_{|C}=f_C<0\end{cases} \tag{3}$$

Où Δ est l'opérateur Laplacien, et les f_C sont des scalaires négatifs à déterminer. Les conditions de flux sont bien connus sous le nom de conditions aux bords de Neumann [SCV14]. De plus l'équation de Laplace avec des conditions de Neumann :

$$\Delta u=0 \quad \text{avec} \quad (\nabla u.\vec{n})_{|\partial\mathbb{S}}=g \tag{4}$$

possède une condition nécessaire sur g pour admettre une solution. Pour obtenir cette dernière on utilise le théorème de Green-Ostrogradski sur une composante connexe X de $\mathbb{S}$:

$$0=\int_X\Delta u=\int_X\nabla\cdot\nabla u=\int_{\partial X}\nabla u.\vec{n}=\int_{\partial X}g$$

Dans notre cas (avec $\partial X_s=X\cap\partial\mathbb{S}_s$ et $\partial X_t=X\cap\partial\mathbb{S}_t$), cela donne :

$$0=l(\partial X_s)+\sum_{C\in\mathcal{C}(\partial X_t)}f_C\,l(C)$$

Où $l(\gamma)$ est la longueur d'une courbe γ. Une solution de l'équation de Laplace (4) est évidement optimale pour notre problème d'optimisation (3) puisque l'intégrale d'une fonction nulle est nulle. Pour ne pas s'embêter on rend égaux les coefficients f_C des boucles de toit C de la composante connexe $X\in\mathcal{C}(\mathbb{S})$ en introduisant la valeur g_X suivante :

$$\forall C\in\mathcal{C}(\partial X_t),\quad f_C=g_X=-\,l(\partial X_s)\,\Big/\,l(\partial X_t)$$

On utilisera alors le champ scalaire défini par la solution de l'équation de Laplace (4) avec g défini comme ci-dessus.

Discrétisation Maintenant on remplace $\mathbb{S}$ par un maillage S avec des faces triangulaires. Comme le Laplacien est un opérateur linéaire, on pose L une matrice qui discrétise le Laplacien sur notre maillage S. Si n est le nombre de sommets, alors L est une matrice carrée de taille $n \times n$. Le champ scalaire est représenté par un vecteur u de taille n donnant la valeur du champ u_i en chaque sommet i. Soit un sommet i de S. On définit la cellule C_i autour de i et dans laquelle l'intégration du Laplacien sera $(Lu)_i$ (voir Figure 10). Dans ce paragraphe on considère un sommet qui n'est pas sur le bord du maillage. Par le théorème de Green-Ostrogradski, on a :

$$(Lu)_i = \int_{C_i} \Delta u = \int_{C_i} \nabla \cdot \nabla u = \int_{\partial C_i} \nabla u . \vec{n} \tag{5}$$

On pose ensuite l_{ij}, la longueur de l'arrête (ij). On note aussi c_{ij}, l'arrête de la cellule C_i qui coupe (ij) comme dans la Figure 10 et $l(c_{ij})$ sa longueur. Finalement on peut reprendre la dernière équation (5) pour obtenir :

$$(Lu)_i = \sum_{j \in \mathcal{N}(i)} \int_{c_{ij}} \nabla u . \vec{n} = \sum_{j \in \mathcal{N}(i)} l(c_{ij}) \frac{u_j - u_i}{l_{ij}} \tag{6}$$

Où $\mathcal{N}(i)$ est l'ensemble des voisins de i.

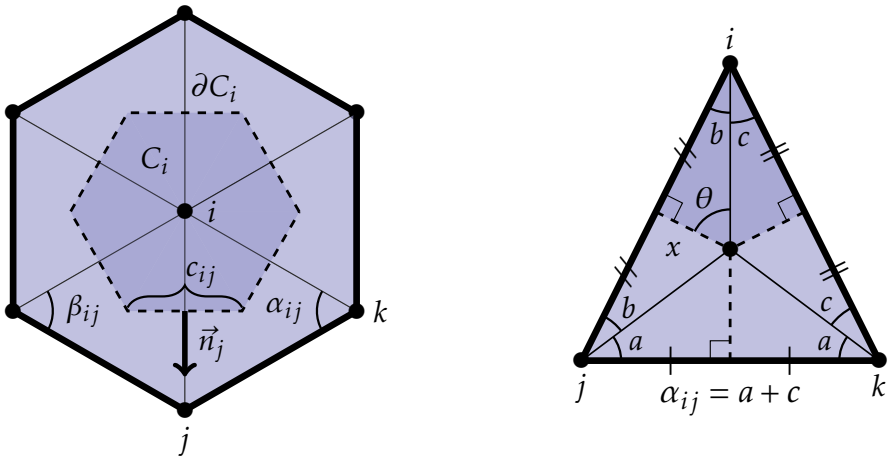


Figure 10 – Discrétisation du Laplacien sur un sommet interne

On doit maintenant déterminer la valeur de $l(c_{ij})$. Pour cela, on note x la longueur de la partie de c_{ij} à l'intérieur du triangle (ijk) comme on peut le voir dans le triangle de la figure précédente. Premièrement, on peut relier x à θ en utilisant de la trigonométrie dans le sous-triangle en haut à gauche :

$$x = \frac{1}{2} l_{ij} \cot \theta$$

Toujours dans le même sous-triangle, en utilisant le fait que la somme des angles dans un triangle vaut π, on obtient :

$$\theta = \frac{\pi}{2} - b$$

En appliquant la même relation dans le triangle entier (ijk), on a :

$$\pi = 2(a+b+c) \quad \Leftrightarrow \quad b = \frac{\pi}{2} - (a+c) = \frac{\pi}{2} - \alpha_{ij}$$

Finalement en injectant l'expression de b dans celle de θ ($\theta = \alpha_{ij}$) que l'on injecte à son tour dans celle de x, on obtient une expression qui dépend de α_{ij} :

$$x = \frac{1}{2} l_{ij} \cot \alpha_{ij}$$

Par symétrie dans le second triangle contenant l'arrête (ij), on a :

$$l(c_{ij}) = \frac{1}{2} l_{ij} (\cot \alpha_{ij} + \cot \beta_{ij})$$

On remplace alors $l(c_{ij})$ dans l'équation (6) pour obtenir :

$$(Lu)_i = \frac{1}{2} \sum_{j \in \mathcal{N}(i)} \Big(\cot \alpha_{ij} + \cot \beta_{ij} \Big) (u_j - u_i) \tag{7}$$

Sur un bord Dans le dernier paragraphe nous avons considéré un sommet i à l'intérieur du maillage. Maintenant on prend i, un sommet du bord. On note c_i la portion du bord ∂M plus proche de i que de n'importe quel autre sommet du bord. Si on se réfère à l'équation (5), nous avons besoin d'intégrer le produit scalaire entre la normale et le gradient de u le long de c_i. Ce produit est fixé par les contraintes de notre problème (3). On note $g_i = \int_{c_i} g$ l'intégral du flux imposé g, le long de c_i. Ce qui nous donne :

$$\int_{C_i} \Delta u = g_i + \frac{1}{2} \sum_{j \in \mathcal{N}(i)} \Big(\cot \alpha_{ij} + \cot \beta_{ij} \Big) (u_j - u_i) = g_i + (Lu)_i \tag{8}$$

Où $\cot \alpha_{ij}$ est nul pour un voisin j tel que aucune face contienne les sommets i suivis de j dans le sens trigonométrique. De la même manière, $\cot \beta_{ij}$ est nul pour un voisin j tel que aucune face contienne les sommets i suivis de j dans le sens horaire.

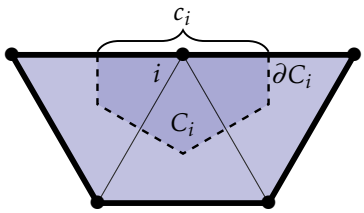


Figure 11 – Discrétisation du Laplacien sur un sommet du bord

Conclusion Les deux derniers paragraphes nous ont permis de construire une matrice L et un vecteur g de la manière suivante (où X_{ij} est la composante connexe de l'arrête (ij)) :

$$L_{ij} = \begin{cases} \frac{1}{2} \Big(\cot \alpha_{ij} + \cot \beta_{ij} \Big) & \text{si } i \neq j \\ -\frac{1}{2} \sum_{j \in \mathcal{N}(i)} \cot \alpha_{ij} + \cot \beta_{ij} & \text{si } i = j \end{cases} \qquad g_i = \frac{1}{2} \sum_{(ij) \in \partial S_s} l_{ij} \; - \; \frac{1}{2} \sum_{(ij) \in \partial M_t} l_{ij} \frac{l\Big(\partial \big(X_{ij}\big)_s\Big)}{l\Big(\partial \big(X_{ij}\big)_t\Big)} \tag{9}$$

Finalement, souhaitant que le Laplacien soit nul, l'intégral du Laplacien sur la cellule C_i doit aussi être nulle. Ainsi, en reprenant l'équation (8), la solution discrète de l'équation de Laplace (4), et du champ de vecteur $\vec{v}$ sont donnés par :

$$Lu = -g \qquad \text{et} \qquad \vec{v} = \nabla u \tag{10}$$

2.2 Réflexion sur les contraintes aux bords

Et Dirichlet ? Bien que le champ proposé ci-dessus ait été correct sur plusieurs modèles, la Figure 12 illustre un problème que nous avons rencontré. On peut voir les murs de ce modèle comme deux grandes bandes l'une au dessus de l'autre reliées par trois tours à droites du modèle. On retrouve aussi des tubes à l'intérieur de ces bandes, mais ils ne nous intéressent pas. On remarque que sur les murs extérieurs, le gradient n'est pas parfaitement vertical comme on aurait aimé qu'il soit. Cela provient du fait que la présence des trois tours supprime une portion de toit sur la bande inférieure et rajoute des demi-cercles de toit sur la bande supérieure. La longueur de toit ajoutée étant plus grande que la longueur supprimée un plus grand flux doit aller en direction de ces trois tours.

Une solution envisagée a été de remplacer nos contraintes de Neumann par des contraintes de Dirichlet. C'est à dire qu'au lieu d'imposer un flux sur les bords on impose la valeur du champ scalaire u. Ainsi on peut imposer la constance de u le long de chaque boucle de bord. Cela permet de forcer la perpendicularité du gradient avec le bord. Et pour que le champ u obtenu ne soit pas nul il suffit de contraindre une boucle de toit à être nulle et de fixer une boucle de sol à la valeur 1. Ce changement de contraintes corrige effectivement le problème obtenu sur ce modèle et permet d'éviter d'envisager un stratagème pour mieux répartir le flux le long des bords.

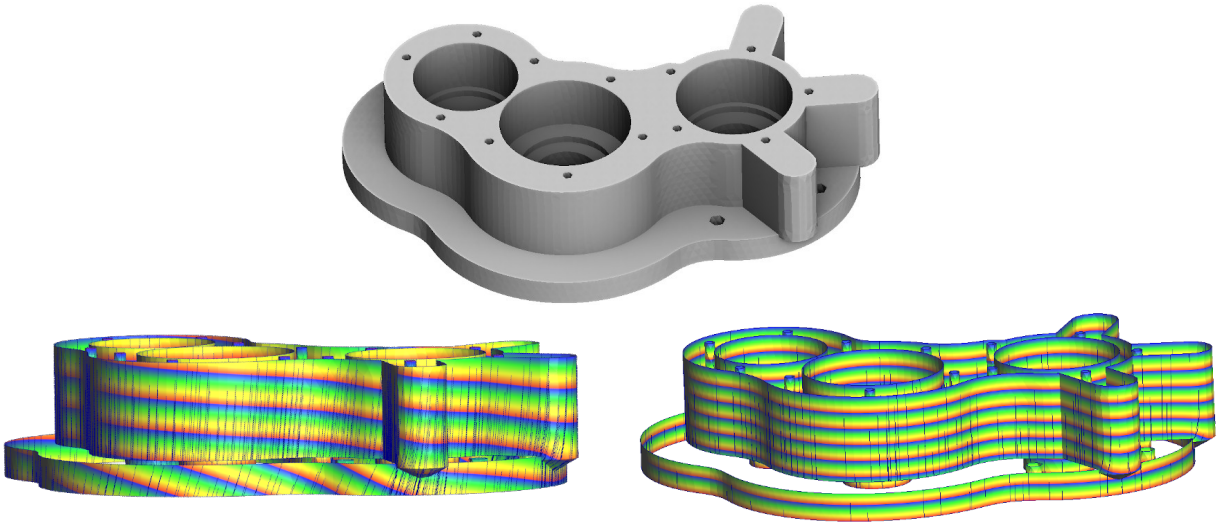


Figure 12 – Comparaison entre le champ obtenu via des contraintes aux bords de Neumann (à gauche) et de Dirichlet (à droite). Le dégradé de couleur représente le champ scalaire. Ainsi une ligne de même couleur est une ligne d'iso-valeur du champ u. Les petits traits bleus sont des *streamlines*.

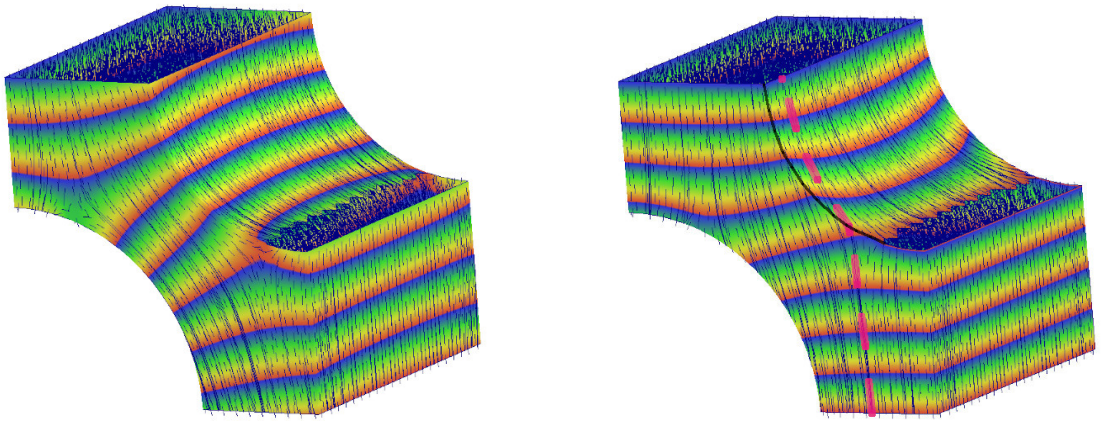


FIGURE 13 – Nouvelle comparaison entre le champ obtenu via des contraintes aux bords de Neumann (à gauche) et de Dirichlet (à droite). Sur la seconde image une ligne d'arrêtes saillantes est marquée en noire et une *streamline* passant par deux faces est marquée en tirets-tillés magenta.

Et un mixe Neumann-Dirichlet ? Finalement en testant sur d'autres modèles on s'aperçoit que les contraintes de Neumann avaient des avantages assez intéressants. En effet, beaucoup de modèles sont construits via des extrusions, c'est à dire qu'une surface 2D est étirée dans une 3ème dimension orthogonale pour former un volume. C'est d'ailleurs sur ces modèles que l'idée des *framfield* 2.5D portent tout leur intérêt. Dans ces modèles les boucles de toit sont simplement des translation (avec éventuellement une rotation) des boucles de sol. Si on note σ l'isométrie qui envoie ∂S_s sur ∂S_t, alors expérimentalement on a tendance à observer qu'un sommet i de ∂S_t se projette sur $\sigma^{-1}(i)$ ce qui est tout à fait souhaitable. Or lorsque la translation possède une composante horizontale (i.e. orthogonale à la direction verticale) alors le champ de vecteurs ne doit pas être orthogonal aux bords du sol comme le contraint les conditions de Dirichlet. La FIGURE 13 illustre ce soucis. On appelle *streamline* une trajectoire dont le vecteur vitesse en un point est donné par le champ $\vec{v}$. Dans cet exemple, avec les conditions de Neumann les *streamlines* restent sur une seule face alors qu'avec les conditions de Dirichlet, certaines *streamlines* passent d'une face à une autre, ce qui n'est pas souhaitable. En effet, tous les points d'une *streamline* seront projetés au même point de ∂S_s. Or, dû aux contraintes imposées par les bords lors de la génération du *framefield* (c.f. équation (1) de la section [1]), les croix doivent potentiellement être différentes entre les deux faces traversées par la *streamline*. Projeter deux sommets de deux faces différentes au même point empêcherait de respecter les contraintes de bord des deux faces simultanément.

Une remarque qui sort du contexte de ce paragraphe est que la direction verticale utilisée pour cet exemple n'est pas la bonne. En effet la direction prise ici nous force à courber les *streamlines* et on se retrouve aussi avec des sols/toits pas très intéressants sur les deux faces courbes. La direction dans la profondeur de l'image aurait été beaucoup mieux. Mais la direction choisie dans l'exemple permet de bien illustrer le problème évoqué.

J'ai aussi pu observé une augmentation du nombre de singularités obtenues à la fin du *pipeline* lorsqu'on utilise des contraintes de Dirichlet. Cela nous amène à considérer un mé-

lange des deux types de contraintes aux bords (Neumann et Dirichlet). Allant sur-contraindre le problème on ne pourra pas nécessairement trouver une solution à l'équation de Laplace en inversant un système linéaire. Il nous faut envisager un minimisation quadratique. On note A_i l'aire de la cellule C_i (représentée sur la Figure 10), de sorte que la valeur moyenne du Laplacien dans cette cellule soit $(Lu+g)_i/A_i$. En effet, on rappelle que L permet d'obtenir l'intégrale du Laplacien sur la cellule avec des contraintes de flux données par g. On minimise désormais :

$$\int_{\mathbb{S}} (\Delta u)^2 = \sum_i A_i \left(\frac{(Lu+g)_i}{A_i} \right)^2 = \sum_i \left(\frac{(Lu+g)_i}{\sqrt{A_i}} \right)^2 = \left\| A^{-1/2}(Lu+g) \right\|^2$$

Où A est la matrice diagonale dont le i-ème coefficient de la diagonale est A_i l'aire de C_i. En ajoutant les contraintes de Dirichlet le problème d'optimisation devient :

$$\min_u \left\| A^{-1/2}(Lu+g) \right\|^2 \qquad \text{t.q.} \quad \forall C \in \mathcal{C}(\partial S),\ \forall (i,j) \in C,\ \ u_i = u_j \tag{11}$$

Hélas il se trouve que ce nouveau champ possède encore des *streamlines* qui passent d'une face à une autre, ce qui nous force à envisager un complément à cette solution.

Contraintes sur les arrêtes saillantes Bien que l'on voit assez clairement ce que l'on évoque lorsqu'on parle de faces, il nous faut tout de même une description formelle. Pour cela on introduit la notion d'arrête saillante (traduction de l'anglais *hard edge*). Une arrête séparant deux triangles de normales $\vec{n}_1$ et $\vec{n}_2$ est dite saillante lorsque l'angle entre les deux normales est grand. Dans notre cas la condition pour qu'une arrête soit saillante est :

$$\left\| \vec{n}_1 \wedge \vec{n}_2 \right\| > 0.7 \tag{12}$$

Ainsi définie, la séparation entre deux faces d'un maillage se fait par des arrêtes saillantes. On souhaite alors qu'aucun flux ne traverse ces arrêtes. Enfin pas toutes. En effet dans la Figure 13 les arrêtes saillantes sont toutes "verticales", mais il peut arriver que deux faces possèdent une jonction horizontale avec un angle souvent assez faible mais suffisamment grand pour satisfaire la condition (12). Un exemple est donné Figure 14. Pour remédier à cela, étant donné que sur certains modèles ont ne peut labelliser une arrête comme horizontale ou verticale qu'à la simple donnée de son vecteur unitaire, on construit d'abord un premier champ $u^{(1)}$ via le programme d'optimisation (11). On en dérive le champ vectoriel :

$$\vec{v}^{(1)} = \nabla u^{(1)} \Big/ \|\nabla u^{(1)}\| \tag{13}$$

Puis on labellise une arrête saillante $e = (ij)$ de vecteur unitaire $\vec{e} = \overrightarrow{ij} / l_{ij}$ comme étant horizontale lorsque le flux la traversant est suffisamment grand. La condition choisie est la suivante :

$$\left| \vec{v}^{(1)} . \vec{e} \right| < 0.7 \tag{14}$$

On calcule ensuite un nouveau champs $u^{(2)}$ en partant de (11) et en y rajoutant des contraintes sur les arrêtes saillantes verticales. On veut que le champ de gradient $v^{(2)} = \nabla u^{(2)} / \|\nabla u_2\|$ soit parallèle aux arrêtes saillantes verticales afin que ces dernières ne soient pas traversé par des *streamlines*. Le champs $u^{(2)}$ étant calculé sur les sommets, on peut l'interpoler sur le maillage

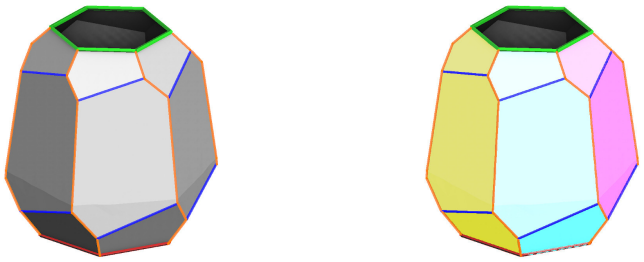


FIGURE 14 – Sur ce maillage, les arrêtes saillantes ont été colorées en bleu pour les horizontales et en orange pour les verticales. L'image de droite représente les clusters obtenus.

entier de sorte à ce que le champ soit linéaire sur chaque triangle. Ainsi le gradient est constant sur chaque triangle. Soit alors un triangle $t = (ijk)$ avec $e = (ij)$ une arrête saillante verticale. On souhaite avoir l'existence d'un scalaire λ tel que $\nabla u_t = \lambda \vec{e}$. On détermine λ via l'équation suivante :

$$u_j = u_i + \overrightarrow{ij}.\nabla u_t = u_i + \lambda l_{ij} \quad \Rightarrow \quad \lambda = \frac{u_j - u_i}{l_{ij}}$$

Puis on obtient la contrainte :

$$u_k = u_i + \overrightarrow{ik}.\nabla u_t = u_i + \left(u_j - u_i\right)\frac{\overrightarrow{ik}.\vec{e}}{l_{ij}}$$

C'est une contrainte linéaire. On pose alors la matrice B qui encode toutes ces contraintes de sorte que l'on veuille $Bu^{(2)} = 0$. Cette matrice possède une ligne par couple (t,e) composé d'un triangle et d'une de ses arrêtes labellisée comme saillante et verticale. On a :

$$(Bu)_{(t,e)} = C^{te}_{(t,e)}\left[u_k + \left(\frac{\overrightarrow{ik}.\vec{e}}{l_{ij}} - 1\right)u_i - \frac{\overrightarrow{ik}.\vec{e}}{l_{ij}}u_j\right] \tag{15}$$

Où $C^{te}_{(t,e)}$ permet d'ajuster l'importance de la contrainte. C'est une valeur proportionnelle à l'inverse de la hauteur du triangle t en prenant pour base e. Réalisant une minimisation quadratique, le programme d'optimisation devient alors :

$$\min_u \left\|A^{-1/2}(Lu + g)\right\|^2 + \|Bu\|^2 \qquad \text{t.q.} \quad \forall C \in \mathcal{C}(\partial S),\ \forall (i,j) \in C,\ \ u_i = u_j \tag{16}$$

Toutefois imposant qu'aucun flux ne passe à travers des arrêtes saillantes verticales, il faut revoir la notion de connexité qui est utilisée pour définir g dans l'équation (9). Auparavant on définissait les chemins comme une suite de faces $(t_1, \ldots, t_N)$ telle que pour tout i, les triangles t_i et t_{i+1} partagent une arrête en commun. Désormais deux triangles consécutifs doivent avoir une arrête en commun qui n'est pas saillante et verticale. Deux triangles appartiennent à la même composante connexe (au même cluster) s'il existe un chemin entre les deux. Sur la FIGURE 14 la surface possède initialement une composante connexe puis avec la nouvelle définition de chemin plus contraignante, la surface se retrouve divisée en 6 clusters. On utilise ensuite ces nouveaux clusters dans la formule de g (9), où le cluster d'une arrête de bord est le cluster de l'unique face qui contient l'arrête. Le champ $u^{(2)}$ est alors de meilleure qualité sur un grand nombre de maillages.

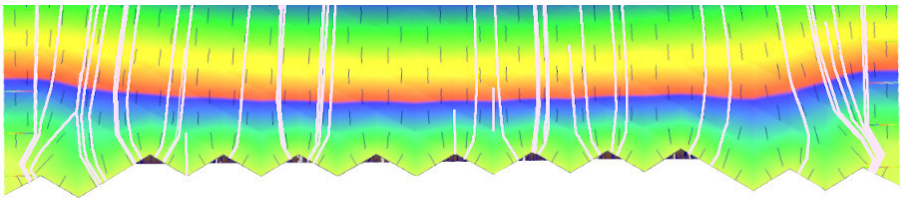


FIGURE 15 – Zoom sur un bord dentelé. Les *streamlines* en blanc se tordent sur la fin.

Sols dentelés Le maillage de la FIGURE 13 s'avère à nouveau utile pour mettre en valeur un défaut. Si on s'intéresse aux deux faces courbes, on remarque qu'un bout de ces faces est labellisé comme horizontal, ce qui se manifeste par un "trou" dans la surface S observée. Or le mur correspondant à la face courbe qui part de cette portion horizontale commence par des arrêtes de sols "dentelées" notamment à cause de la presque horizontalité de ce mur. Les contraintes de Dirichlet qui imposent l'orthogonalité du gradient par rapport à ce bord auront tendance à tordre les *streamlines* comme on peut le voir FIGURE 15. Pour éviter cela on détecte ces arrêtes de sols dentelés et on y supprime les contraintes de Dirichlet. A noter qu'on enlève pas les contraintes sur toute la boucle de bord mais uniquement sur les arrêtes nécessaires. Cette modification corrige le soucis comme on peut le voir, si on zoom sur la FIGURE 16.

Résultat final Pour remettre de l'ordre dans tout ça, on résume les étapes :

- Calcul des composantes connexes $\mathcal{C}(\partial S)$.
- Calcul de la matrice L et du vecteur g (9).
- Calcul du champ scalaire $u^{(1)}$ (11) et du champ vectoriel $\vec{v}^{(1)}$ (13).
- Détection des arrêtes saillantes (12) et verticales (14).
- Calcul de la matrice B (15).
- Mise à jour des clusters et mise à jour de g.
- Calcul du champ scalaire $u^{(2)}$ (16) et vectoriel $\vec{v}^{(2)}$

La FIGURE 16 montre les nouveaux champs obtenus sur les deux modèles qui ont illustré les problèmes précédents. Les soucis ont disparus, il n'y a plus de champs de travers, plus de *streamlines* qui passent d'une face à une autre ni de *streamlines* tordues proche des sols dentelés.

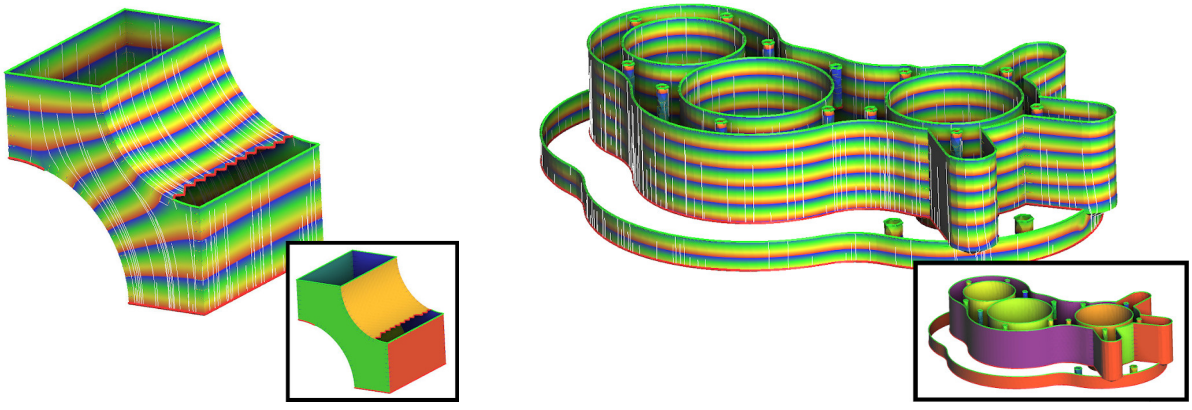


FIGURE 16 – Les nouveaux champs scalaires u obtenus sur les deux exemples. Les petits cadres représentent également les nouveaux clusters (une couleur par cluster).

2.3 Projection

Une fois ce champ $\vec{v}$ calculé, un sommet i du mur S est projeté sur le point d'arrivé de la *streamline* qui part de i. On note $p_0(i)$ cette projection et $\lambda(i)$ la longueur de la *streamline* allant de i à $p_0(i)$. Dans la section suivante, on déformera le maillage de sorte à ce que la projection se fasse non plus en suivant des *streamlines* mais en se déplaçant verticalement vers le sol. Dans cette déformation, la nouvelle position d'un sommet i du mur sera alors définie par $T_0(i) = p_0(i) + \lambda(i)\vec{z}_0$ afin que la projection de i soit obtenu en partant de $T_0(i)$ et en se déplaçant dans la direction $-\vec{z}_0$ jusqu'à atteindre le sol. Cela engendre des nouveaux murs verticaux. La déformation donnera ainsi, un nouveau maillage construit via une extrusion du sol. La Figure 17 représente cette projection et la déformation qui en découle.

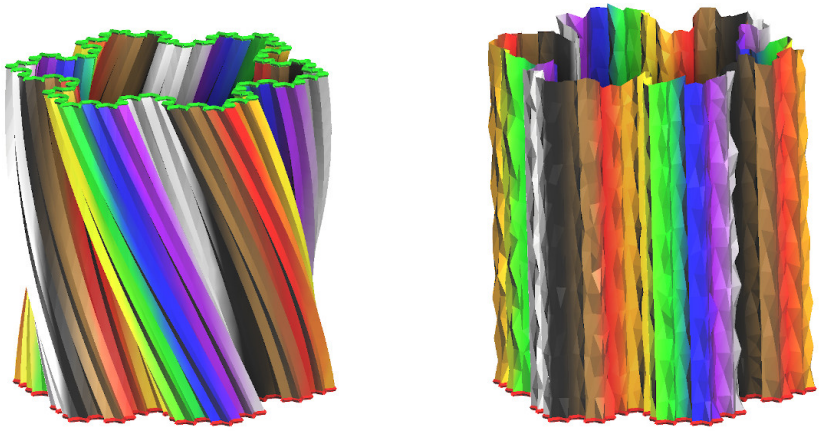


Figure 17 – Projection des murs sur le sol. Une couleur est donnée à chaque sommet du sol. La couleur d'un sommet i du mur est alors la couleur au point du sol $p_0(i)$ sur lequel il est projeté. Le maillage initial S a gauche est ensuite redressé en $T_0(S)$ à droite.

3 Déformation et projection du volume

Pour définir la projection de l'intérieur du maillage sur le sol, l'idée que nous avons eu consiste à déformer l'intérieur du maillage en s'appuyant sur la déformation des murs de la section précédente [2.3]. La projection d'un sommet i sera alors définie comme la première intersection avec une surface de sol de la demie droite partant de i et de direction $-\vec{z}_0$. Il existe déjà de nombreux travaux qui ont étudié des déformations de maillages. On peut penser à la déformation *As Rigid As Possible* datant de 2007 [SA07], ou encore une de ses amélioration plus récentes [LG15]. Comme déjà expliqué, nous avons d'abord voulu mettre en place un *pipeline* complet et dans un second temps améliorer les points faibles de notre algorithme de génération de maillage hexaédrique. Nous avons alors dans un premier temps opté pour une solution plus simple que ARAP. Ce dernier n'aura finalement pas été implémenté étant donné que nous avons fini par nous orienter vers de l'amélioration de *framefield* à la place de la génération de maillage *from scratch*.

On représente la déformation T par des translations locales $t(i) \in \mathbb{R}^3$ en chaque sommet i, en oubliant pas les contraintes d'égalité entre T et T_0 sur les murs. C'est à dire :

$$T(i) = i + t(i) \qquad \text{avec } \forall i \in S,\ t(i) = T_0(i) - i$$

On cherche alors à définir ce champ de vecteurs t le plus lisse possible. Pour se faire on peut minimiser le carré du gradient de t. Dans un domaine continue, cela reviendrait à :

$$t = \arg\min_t \int_{\mathbb{M}} \left(\|\nabla t_x\|^2 + \|\nabla t_y\|^2 + \|\nabla t_z\|^2 \right) \qquad \text{t.q. } t_{|S} = T_0 - id_S \tag{17}$$

Le principe de Dirichlet nous donne l'équivalence entre cette minimisation et l'équation de Laplace avec contraintes de Dirichlet :

$$\Delta t_x = 0 \qquad \Delta t_y = 0 \qquad \Delta t_z = 0 \qquad \text{avec } t_{|S} = T_0 - id_S$$

Comme dans la section [2.1] on peut définir une matrice L_{3D} représentant une discrétisation du Laplacien associé à un maillage tétraédrique [Cra19]. Il suffit alors de résoudre les trois équations linéaires :

$$L_{3D} t_x = 0 \qquad L_{3D} t_y = 0 \qquad L_{3D} t_z = 0 \qquad \text{avec } \forall i \in S,\ t(i) = T_0(i) - i$$

Bien qu'ayant implémenté le calcul de cette matrice L_{3D} plus tardivement dans ce stage, j'ai opté pour un calcul de t en faisant une minimisation quadratique d'une discrétisation de l'équation (17). En effet, voulant remplacer par la suite cette déformation par une transformation *as rigid as possible* [SA07], j'ai décidé de faire le calcul le plus simple possible pour T. La minimisation effectuée est alors la suivante :

$$t = \arg\min_t \sum_{(ij) \in M} \frac{1}{l_{ij}} \|t(j) - t(i)\|^2 \qquad \text{t.q. } \forall i \in S,\ t(i) = T_0(i) - i$$

Il est à noter que ce modèle de déformation est très limité. Par exemple pour des torsions comme on peut voir sur la Figure 17, le vecteur de translation t tourne rapidement le long de la paroi, ce qui complique énormément la création d'un champ lisse. Dans ce cas on peut observer plusieurs tétraèdres qui s'inversent après transformation. Toutefois voici un exemple de déformation obtenue sur un modèle qui se prête bien à la déformation implémentée Figure 18.

Projection Pour la projection de l'intérieur du volume sur le sol, comme expliqué en début de section, elle se fait en partant d'un point du volume et en avançant dans la direction $-\vec{z}_0$ jusqu'à rencontrer un sol. Afin d'utiliser des outils d'intégration de *framefield* écrit par l'équipe PIXEL, le *framefield* sera échantillonné sur les tétraèdres du maillage et non les sommets. Pour une cellule tétraédrique c, la projection $p(c)$ de celle-ci, est la projection de son barycentre $\bar{c}$. C'est à dire l'intersection du premier sol rencontré avec la demie-droite partant de $\bar{c}$ et de direction $-\vec{z}_0$. La position de cette projection $p(c)$ est donnée par un couple (τ, Γ) où $\tau = (ijk)$ est un triangle de sol et $\Gamma = (\alpha, \beta, \gamma)$ est un triplet de réels représentant les coordonnées barycentriques de $p(c)$ en fonction des sommets du triangle τ. C'est à dire :

$$p(c) = \alpha i + \beta j + \gamma k \qquad \text{avec } \alpha + \beta + \gamma = 1$$

En réalité pour que tous ces calculs se fassent rapidement il est préférable de partir des triangles de sol et de remonter en construisant un chemin à travers les tétraèdres afin de trouver toutes les cellules se projetant dans ce triangle. Cette partie est algorithmiquement intéressante mais je ne m'attarderai pas dessus.

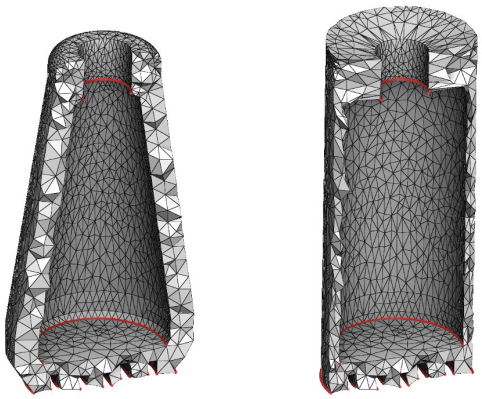


FIGURE 18 – A gauche un maillage et sa déformation à droite.

Discussion Le *pipeline* pour obtenir cette projection peut sembler long et fastidieux. Moi même je ne suis pas totalement convaincu. En effet, le champ scalaire calculé sur les murs [section 2] peut tout à fait être défini sur le maillage entier, avec approximativement les mêmes contraintes (il n'y a plus d'arrêtes saillantes et l'intégration du flux ne se fait plus sur des arrêtes mais sur des triangles) et en remplaçant simplement la matrice L qui intègre le Laplacien sur une surface par la matrice L_{3D} qui intègre le Laplacien sur un volume [Cra19]. La projection serait à nouveau définie par la première intersection du sol avec une *streamline* partant d'un point du maillage et guidée par le gradient du champ scalaire. Il n'y aurait alors pas de déformation à calculer. J'ai implémenté cette méthode, mais ne l'ai probablement pas assez testé. Je n'y ai pas vu d'avantages notables. De plus calculer des *streamlines* partant de chaque barycentre de tétraèdre est plus long que la projection verticale dans le modèle déformé car si l'on veut être précis avec les *streamlines* il est difficile d'optimiser leurs calculs. Nicolas qui avait envisagé de commencer à travailler sur les murs semblait assez convaincu par le *pipeline* présenté dans ce rapport et j'ai fait confiance à son expérience là dessus.

4 *framefield* 2D

4.1 Création d'une base orthonormale en chaque tétraèdre

Comme illustré FIGURE 9 avec le cylindre courbé qui effectue un quart de tour, il est possible que la direction verticale évolue au sein du maillage. Comme cette direction sera un des vecteur de nos croix dans le *framefield* 3D, il nous faut calculer ce champ de vecteur que l'on note $\vec{z}$. On l'échantillonne en chaque cellule. Ce champ est contraint par les bords du maillage. On se sert à nouveau du champ $\vec{v}$ calculé sur les murs. Voici les 3 types de contraintes :

$$\vec{z} = -\vec{v} \text{ sur } \partial M_m \qquad \vec{z} = -\vec{n} \text{ sur } \partial M_s \qquad \vec{z} = \vec{n} \text{ sur } \partial M_t \tag{18}$$

Où $\vec{n}$ est le vecteur normal dirigé vers l'extérieur. A l'intérieur du maillage on souhaite que le champ $\vec{z}$ soit le plus lisse possible et unitaire. La contrainte d'unitarité n'étant pas linéaire, on procède en plusieurs optimisations successives. L'initialisation est donnée par :

$$\vec{z}^{(0)} = \arg\min_{\vec{z}} \sum_{c \sim c'} \frac{1}{\|\bar{c} - \bar{c}'\|} \|\vec{z}_c - \vec{z}_{c'}\|^2 \qquad \text{t.q. (18)}$$

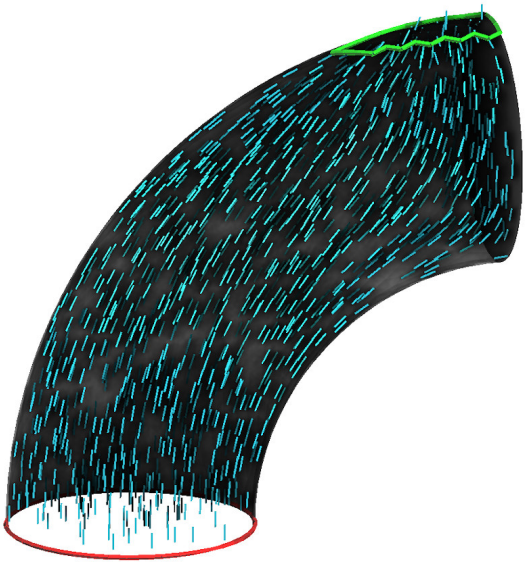


FIGURE 19 – Représentation du vecteur $\vec{z}$ dans le cylindre tordu.

Puis on calcule le champ à l'étape $t+1$ en se servant de celui calculé à l'étape t via :

$$\vec{z}^{(t+1)} = \arg\min_{\vec{z}} \sum_{c\sim c'} \frac{1}{\|\bar{c}-\bar{c}'\|}\|\vec{z}_c - \vec{z}_{c'}\|^2 \; + \; C^{te}(t) \sum_{c} \left(\frac{\vec{z}_c \cdot \vec{z}_c^{(t)}}{\left\|\vec{z}_c^{(t)}\right\|} - 1 \right)^2 \qquad \text{t.q. (18)}$$

Où $c \sim c'$ signifie que les cellules (tétraèdres) c et c' sont adjacentes et $C^{te}(t)$ est une fonction croissante de t. Moins d'une dizaine d'itérations suffisent généralement à atteindre la convergence. La FIGURE 19 montre le résultat obtenu sur le maillage évoqué en début de paragraphe. Le champ se comporte comme voulu. A remarquer qu'arrivé en haut du maillage le champ arrête de suivre la trajectoire du cylindre pour venir s'orienter vers le toit qui a été déterminé précédemment bien qu'il ne correspondent pas nécessairement à ce que l'on aurait aimé avoir.

Enfin il nous faut déterminer une base du plan orthogonale à $\vec{z}$ en chaque cellule afin de pouvoir paramétrer les croix du *framefield* 2D avec un angle. Comme un seul angle sera utilisé pour toutes les croix des cellules qui se projettent au même point, il n'est pas question de choisir des bases aléatoires, il faut une certaine cohérence. Pour se faire on choisi arbitrairement deux vecteur $\vec{x}_0$ et $\vec{y}_0$ afin que le triplet $(\vec{x}_0, \vec{y}_0, \vec{z}_0)$ forme une base orthonormée de $\mathbb{R}^3$. Ensuite on définira une base $(\vec{x}, \vec{y}, \vec{z})$ en chaque cellule. Cette base peut être construite en appliquant la "plus petite" rotation qui envoie $\vec{z}_0$ sur $\vec{z}$. L'axe de cette rotation est définie par :

$$\vec{r} = \vec{z}_0 \wedge \vec{z}$$

Puis la matrice de rotation est donnée par :

$$R = rr^{\mathsf{T}} + zz_0^{\mathsf{T}} + (r \wedge z)(r \wedge z_0)^{\mathsf{T}}$$

Où les vecteurs sont pris comme des vecteurs colonnes et x^{T} est la transposé de x. Finalement on obtient notre base par cellule via :

$$(\vec{x}, \vec{y}, \vec{z}) = R \cdot (\vec{x}_0, \vec{y}_0, \vec{z}_0)$$

4.2 Génération du *framefield* 2D

Représentation des croix Cette partie a déjà bien été étudié dans [RS15]. Premièrement il nous faut savoir comme représenter une croix constituée de 4 vecteurs unitaires invariante par rotation de $\frac{\pi}{2}$. Si on prend l'angle orienté θ entre le premier vecteur de la base, $\vec{x}$, et un des 4 vecteurs de la croix on se rend compte qu'il n'y a pas unicité de la représentation puisque le choix d'un autre vecteur de la croix donne un angle différent. En revanche avec l'invariance par rotation de $\frac{\pi}{2}$, multiplier l'angle par 4 nous donne bien une représentation unique (4θ est identique modulo 2π quel que soit l'angle θ choisi parmi les 4 vecteurs de la croix). Un angle étant pris modulo 2π il est difficile d'effectuer une optimisation. Encore une fois un soucis de non unicité de la représentation apparaît. On peut remédier à cela en considérant les deux coordonnées du vecteur $(a, b) = (\cos(4\theta), \sin(4\theta))$. Maintenant, étudions un peu plus la possibilité de considérer ce vecteur.

Pour lisser le *framefield*, afin d'éviter les distorsions, il va nous falloir une distance entre les croix. Cette distance doit être invariante par rotation :

$$d(\theta_1 + \alpha, \theta_2 + \alpha) = d(\theta_1, \theta_2)$$

La distance est alors entièrement définie par la donnée de $d(0, \theta)$. Avec les symétries de la croix on doit aussi avoir les deux égalités suivantes :

$$d(0, \theta + \pi/2) = d(0, -\theta) = d(0, \theta)$$

Enfin, avec les propriétés précédentes, la fonction $\theta \mapsto d(0, \theta)$ est complètement déterminée par sa valeur sur $[0, \pi/4]$ sur laquelle elle doit être croissante. Il se trouve que la distance :

$$d(\theta_1, \theta_2) = \|(\cos(4\theta_2), \sin(4\theta_2)) - (\cos(4\theta_1), \sin(4\theta_1))\|_2$$

vérifie toutes ces propriétés. C'est donc cette représentation des croix sous forme de vecteur qui est utilisée. Chaque croix est représentée par deux variables $a = \cos(4\theta)$ et $b = \sin(4\theta)$ (Figure20).

Ces croix sont échantillonnées sur les sommets du sol. Pour chaque sommet $i \in \partial M_s$ on dispose alors de deux variables $a_i = \cos(4\theta_i)$ et $b_i = \sin(4\theta_i)$. Pour une cellule c du maillage, on

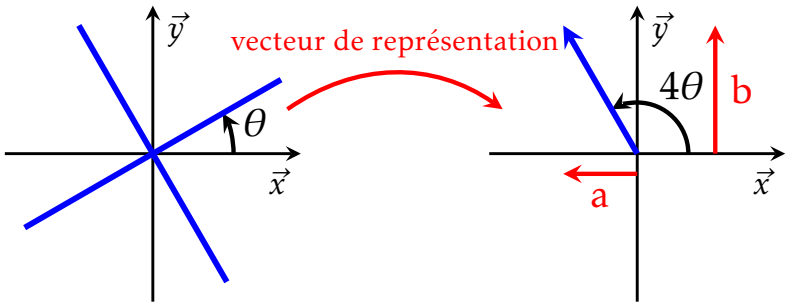


Figure 20 – Représentation des croix

rappelle que le barycentre est projeté au point $p(c)$ défini par les coordonnées barycentriques $\Gamma_c = (\alpha_c, \beta_c, \gamma_c)$ dans le triangle $\tau_c = (i_c, j_c, k_c)$. Les variables a et b étant échantillonnées sur les sommets de τ_c, on suppose qu'elles évoluent linéairement à l'intérieur du triangle. La croix en c est donc donnée par le vecteur de représentation (a_c, b_c) défini par :

$$a_c = \alpha_c a_{i_c} + \beta_c a_{j_c} + \gamma_c a_{k_c} \qquad b_c = \alpha_c b_{i_c} + \beta_c b_{j_c} + \gamma_c b_{k_c} \tag{19}$$

Contraintes de bords En revanche des contraintes provenant de tout le volume peuvent être introduites via la projection. Pour que le *framefield* soit conforme au bord, il faut qu'un des vecteurs de chaque croix soit orthogonal au bord. Pour les toits et les sols, cette contrainte est déjà vérifiée par $\vec{z}$ (18). Ainsi, seuls les murs sont contraignant pour notre *framefield*. Soit un triangle t de mur ayant une normale dirigée vers l'extérieur $\vec{n}_t$. Ce triangle t est la face d'un seul tétraèdre c. On rappelle qu'avec l'équation (18), on doit avoir $\vec{z_c} \simeq -\vec{v}_t$ où $\vec{v}_t$ appartient à t et est donc orthogonal à $\vec{n}_t$. D'où $\vec{z_c}.\vec{n}_t \simeq 0$. $\vec{n}_t$ étant pris unitaire, on a donc :

$$\vec{n}_t \simeq \cos(\phi_t)\,\vec{x_c} + \sin(\phi_t)\,\vec{y_c} \qquad \text{avec } \phi_t = \operatorname{atan2}(\vec{y_c}.\vec{n}_t,\ \vec{x_c}.\vec{n}_t)$$

Où $\operatorname{atan2}(b, a)$ donne l'angle du vecteur (a, b) dans $]-\pi, \pi]$. Ensuite, pour qu'un des vecteurs de la croix soit égal à $\vec{n}_c$ il faut la congruence $4\theta_c \equiv 4\phi_t\,[2\pi]$, ce qui se traduit par :

$$a_c = \cos(4\phi_t) \qquad \text{et} \qquad b_c = \sin(4\phi_t)$$

Comme de multiples triangles de bords peuvent amener à des contraintes légèrement contradictoires on ne peut pas imposer les égalités ci-dessus. On procède donc à la minimisation de $d(\theta_c, \phi_t)^2$. C'est à dire :

$$\sum_{(t,c)\in\partial M_m} (a_c - \cos(4\phi_t))^2 + (b_c - \sin(4\phi_t))^2$$

Lissage On aimerait bien que les croix de deux cellules/tétraèdres adjacents soient proches. En revanche, minimiser la distance entre toutes les paires de cellules adjacentes peut être assez redondant puisque plusieurs cellules se projettent dans les mêmes triangles de sol. Le lissage se fait alors simplement en minimisant la distance des croix entre deux sommets de sol reliés par une arrête. On minimise ainsi :

$$\sum_{(ij)\in\partial M_s} \frac{1}{l_{ij}} \left((a_j - a_i)^2 + (b_j - b_i)^2 \right)$$

Toutefois cela n'est pas suffisant. En effet le sol n'est pas forcément connexe et il se peut que deux cellules adjacentes se projettent sur deux composantes connexes de sol différentes. Dans ce cas la minimisation précédente ne garantie pas le lissage entre ces deux cellules. On note $\sim_{\mathcal{C}}$ la relation d'appartenance à la même composante connexe. On rajoute donc le terme suivant à minimiser :

$$\sum_{\substack{c\sim c' \\ p(c)\sim_{\mathcal{C}} p(c')}} \frac{1}{l_{cc'}} \left((a_{c'} - a_c)^2 + (b_{c'} - b_c)^2 \right)$$

Où les termes a_c et b_c sont définis comme dans (19) et $l_{cc'}$ est la distance entre les barycentres des cellules c et c'.

Contrainte unitaire Enfin chaque paire de variables $a_i = \cos(4\theta_i)$ et $b_i = \sin(4\theta_i)$ est reliée par une équation non linéaire :

$$a_i^2 + b_i^2 = 1$$

Utilisant un solveur linéaire on ne peux pas directement minimiser $(a_i^2 + b_i^2 - 1)^2$. On procède alors en plusieurs itérations. A l'itération s on calcul les valeurs $a_i^{(s)}$ et $b_i^{(s)}$. Puis à l'itération $s+1$ on minimise :

$$\sum_{i \in \partial M_s} \left(a_i^{(s+1)} a_i^{(s)} + b_i^{(s+1)} b_i^{(s)} - 1 \right)^2$$

Résumé Finalement, on obtient le *framefield* en optimisant les deux valeurs a et b initialisées à :

$$a^{(0)}, b^{(0)} = \arg\min_{a,b} \left\{ \begin{array}{l} C_1 \displaystyle\sum_{(t,c) \in \partial M_m} (a_c - \cos(4\phi_t))^2 + (b_c - \sin(4\phi_t))^2 \\ \quad + \displaystyle\sum_{(ij) \in \partial M_s} \frac{1}{l_{ij}} \left((a_j - a_i)^2 + (b_j - b_i)^2 \right) \quad + \displaystyle\sum_{\substack{c \sim c' \\ p(c) \sim_{\mathcal{C}} p(c')}} \frac{1}{l_{cc'}} \left((a_{c'} - a_c)^2 + (b_{c'} - b_c)^2 \right) \end{array} \right. \tag{20}$$

Où C_1 est une constante déterminant l'importance des conditions au bords (par exemple 100 divisé par la valeur moyenne des distances entres deux sommets adjacents du maillage). Puis on procède à une cinquantaine d'itérations mettant à jour a et b via l'équation :

$$a^{(s+1)}, b^{(s+1)} = \arg\min_{a,b} \left\{ \begin{array}{l} C_1 \displaystyle\sum_{(t,c) \in \partial M_m} (a_c - \cos(4\phi_t))^2 + (b_c - \sin(4\phi_t))^2 \\ \quad + \displaystyle\sum_{(ij) \in \partial M_s} \frac{1}{l_{ij}} \left((a_j - a_i)^2 + (b_j - b_i)^2 \right) \quad + \displaystyle\sum_{\substack{c \sim c' \\ p(c) \sim_{\mathcal{C}} p(c')}} \frac{1}{l_{cc'}} \left((a_{c'} - a_c)^2 + (b_{c'} - b_c)^2 \right) \\ \quad + C_2(s) \displaystyle\sum_{i \in \partial M_s} \left(a_i^{(s+1)} a_i^{(s)} + b_i^{(s+1)} b_i^{(s)} - 1 \right)^2 \end{array} \right. \tag{21}$$

Où C_2 est une seconde constante croissante au cours des itérations allant jusqu'à une valeur proche de C_1.

Résultats Finalement en considérant l'angle d'un des vecteurs $\theta_c = \frac{1}{4} \cdot \text{atan2}(b_c, a_c)$ du plan orthogonal à $\vec{z}$, on défini la croix de la cellule c par l'ensemble des trois vecteurs (et de leurs opposés) suivant :

$$\{ \cos(\theta_c)\vec{x}_c + \sin(\theta_c)\vec{y}_c, \quad -\sin(\theta_c)\vec{x}_c + \cos(\theta_c)\vec{y}_c, \quad \vec{z}_c \}$$

Afficher des milliers de croix n'est pas très satisfaisant à visualiser, la Figure 21 propose alors des représentations alternatives du *framfield* sur le maillage *notch* souvent utilisé en contre-exemple des anciennes méthodes de génération de *framefield*. On observe la présence des singularités souhaitées (voir Figure 6). Elles suivent bien la direction stable.

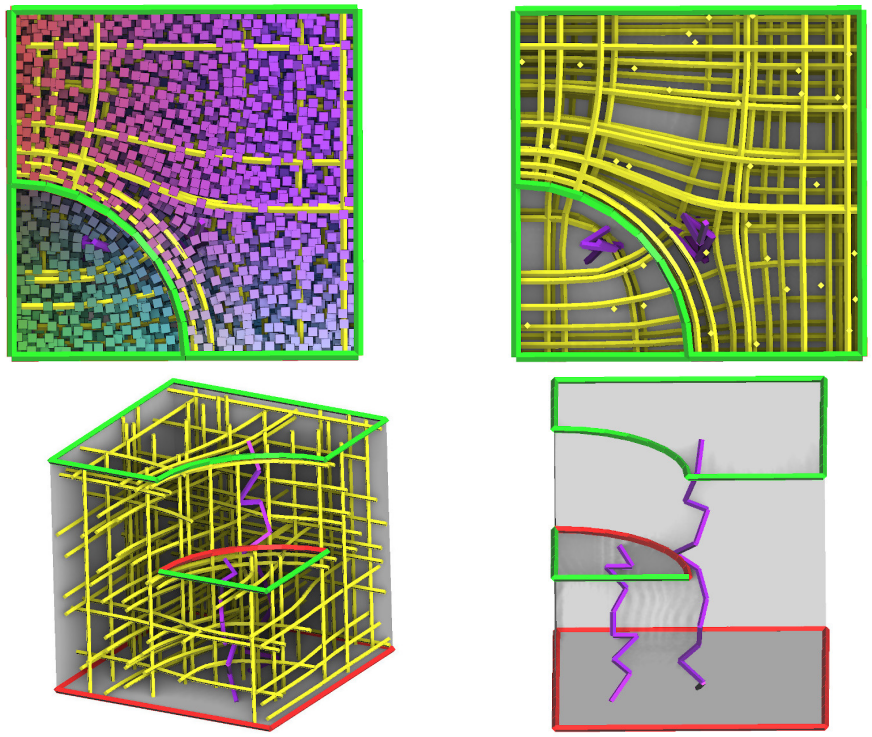


FIGURE 21 – Représentations du *framefield* sur le modèle *notch*. Les traits jaunes représentes des *streamlines*. Les deux singularités sont en violet. L'image en haut à gauche contient des cubes dont les faces sont alignés avec les direction du *framfield*.

5 Fin du *pipeline* et résultats

Pour cette partie, tous les outils nécessaires ont déjà été implémentés par l'équipe PIXEL. J'ai juste eu à utiliser leur implémentation de *Cube Cover* [NRP11] ainsi que *Hexex* [LBK16] qui améliore la phase d'extraction des hexaèdres. La FIGURE 22 montre trois exemples de résultats satisfaisants obtenus. En réalité trop peu de maillages sont correctement maillés (seule une dizaine sur une cinquantaine de maillages à disposition). Le plus grand frein est le fait que l'on utilise une seule direction stable pour tout le maillage. Ce n'est absolument pas suffisant. Si l'on considère l'exemple de la FIGURE 23 dans laquelle la direction verticale est choisie de bas en haut, le grand tube est correctement maillé car les deux singularités de valence 5 qu'il implique sont dans le sens de la verticalité. En revanche le second petit tube sur lequel un zoom est effectué implique des singularités dont la somme des indices est 4 (par exemples 4 singularités de valence 5) et dont la direction est horizontale (le long du tube). Ces singularités ne peuvent pas être construites avec l'algorithme décrit jusque là qui ne peut créer que des singularités verticales. Il est nécessaire d'avoir une seconde direction de verticalité $\vec{z}_0$ pour cette partie du maillage.

Nous avons eu toute une phase de réflexion pour remédier à ce problème en commençant à envisager de multiples solutions. Ces solutions consistaient en la segmentation du bord du maillage en plusieurs clusters puis en une phase de tests pour savoir s'il était possible de tirer

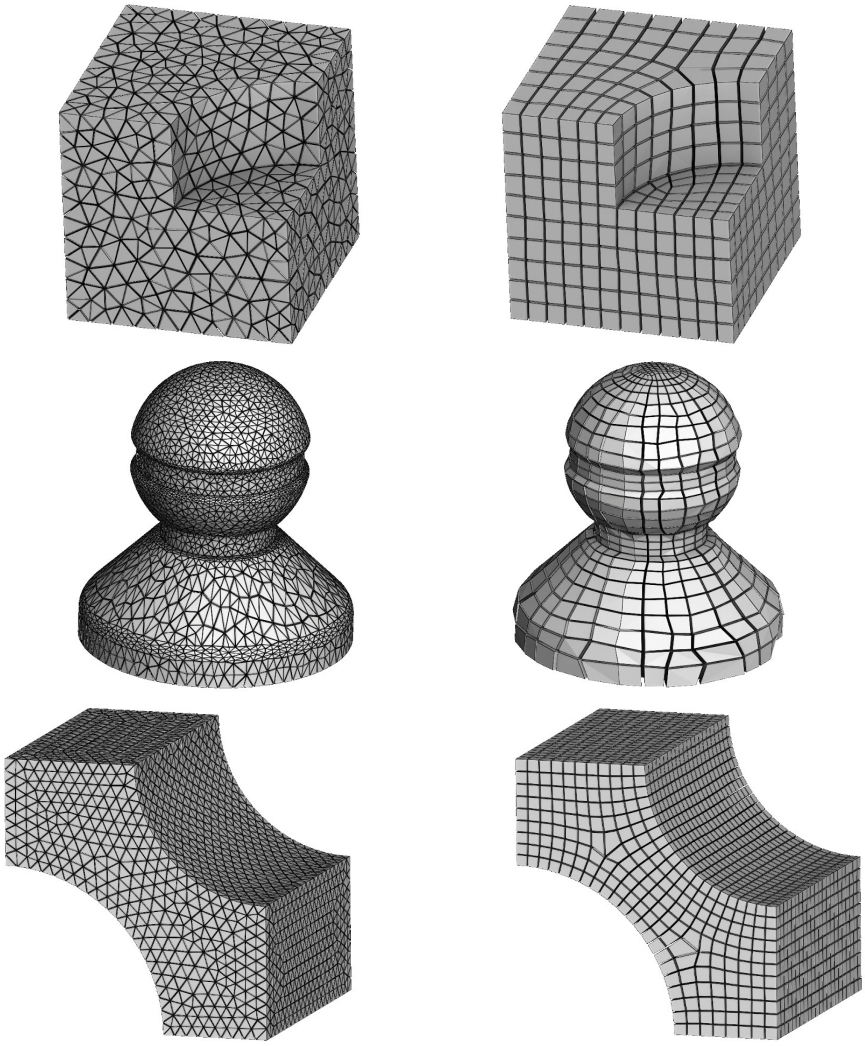


FIGURE 22 – Quelques résultats affichés par paires de maillage tétraédrique initiale et maillage hexaédrique obtenu.

une direction stable d'une face impliquant une singularité (via la formule (2)) vers une autre. Finalement Nicolas s'est rendu compte que ce que l'on voulait, était simplement un premier *framefield* qui donnerait les directions stables. Les *framefield* 2.5D pourraient ensuite intervenir pour corriger localement certaines singularités qui ne suivent pas leur direction stable. Sans forcément corriger des singularités non valides, rectifier localement des *framfields* pourrait permettre de replacer les singularités plus loin des bords, l'optimisation d'un *framfield* 2D se faisant mieux que directement en 3D.

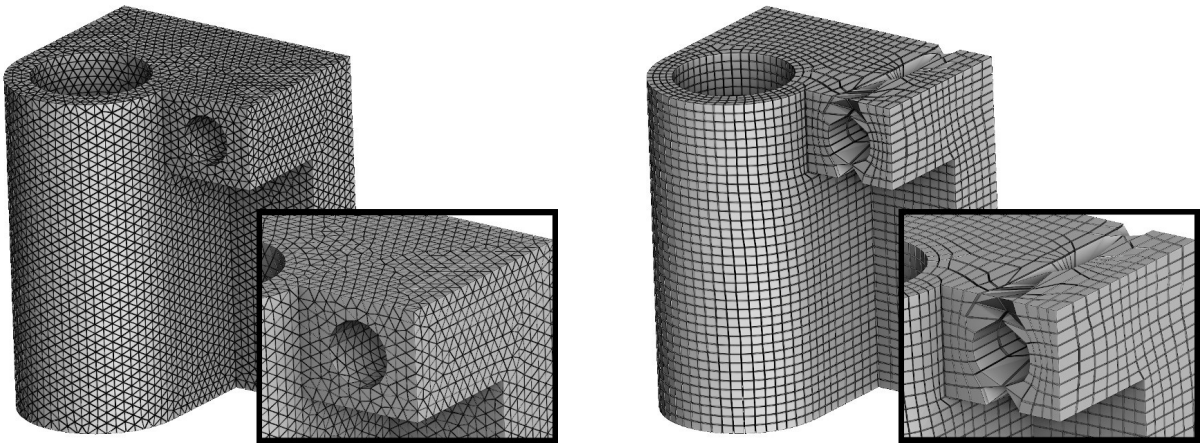


Figure 23 – Exemple de mauvaise hexaédrisation.

Deuxième partie
Correction de singularités

Dans cette partie on part d'un maillage tétraédrique accompagné d'un premier *framefield* généré via d'anciennes méthodes [RS15]. Le but est ensuite de recalculer le champ de croix en utilisant de la 2.5D autour des singularités en prenant pour direction verticale, la direction stable des singularités. Cette méthode permettrait entre autres de corriger des singularités non valides (qui ne suivent pas la direction stable).

Pour se faire il nous faut définir ce que l'on entend par "autour". Ce "autour", sera un sous-ensemble/cluster connexe de tétraèdres du maillage en entrée. Le bord de ce sous-ensemble contiendra alors des triangles de bord du maillage mais aussi des triangles internes adjacents à un tétraèdre n'appartenant pas au sous-ensemble. Dans le premier cas, seul une direction du *framefield* est fixée par la direction normale au triangle. Dans le second cas, le *framefield* est entièrement fixé (en se laissant une petite marge d'erreur car on souhaite obtenir un champ de croix lisse et non constant) par la croix associée au tétraèdre de l'autre côté du triangle. De plus, de par la structure d'un *framfield* 2.5D, toutes les croix se projetant au même point se retrouvent à leur tour contraintes. Ainsi si l'on souhaite pouvoir modifier le *framefield*, pour un tétraèdre c à l'intérieur du cluster, il faut que tous les autre tétraèdres se projetant au même point appartiennent aussi au cluster. C'est à dire que tous les tétraèdres rencontrés par la *streamline* qui suit la direction verticale et qui passe par c doivent aussi appartenir au cluster. En effet si un de ces tétraèdres c' n'appartient pas au cluster alors une de ces faces est adjacente avec un autre tétraèdre de la *streamline*, c''. La croix de c'' est alors contrainte par la croix de c'. Puis c'' et c étant sur la même *streamline*, la croix de c se retrouve elle aussi contrainte. Les clusters que l'on doit construire doivent alors ressembler à des cylindres déformés dont la direction est la direction stable d'une singularité.

Pour construire ces clusters, en pratique nous faisons grossir des cylindres en partant des singularités jusqu'à rencontrer des contraintes. Avec cette idée de base, nous avons testé de nom-

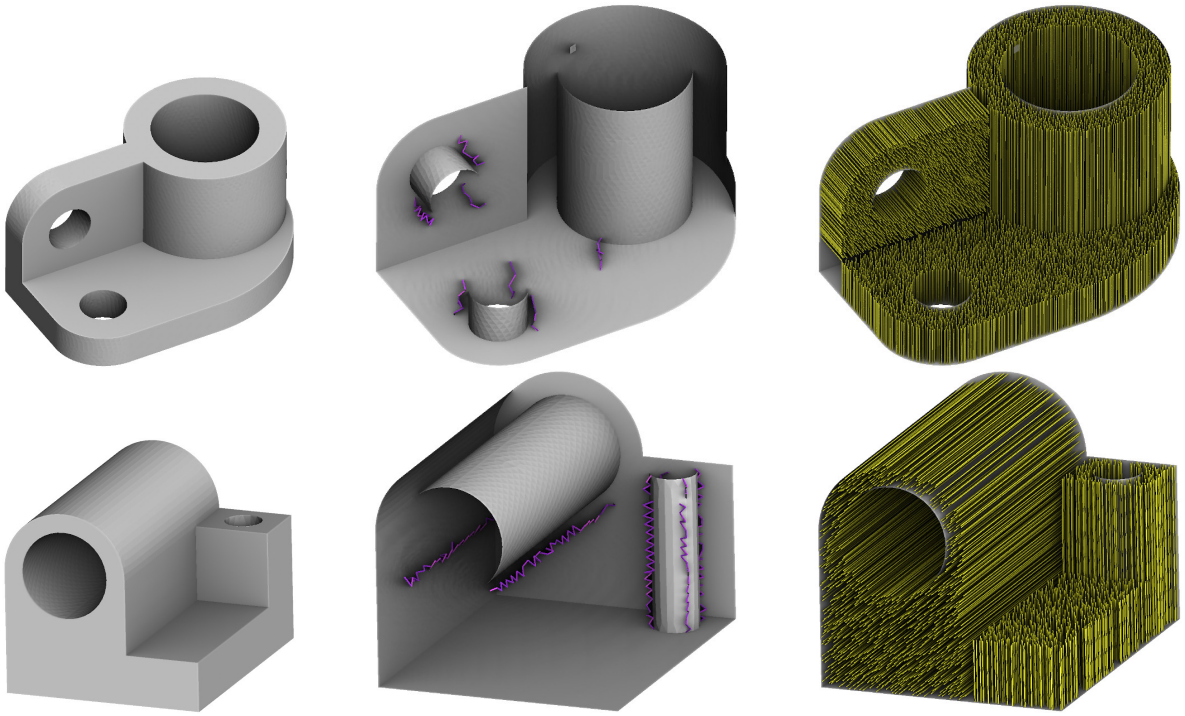


FIGURE 24 – Résultats d'une ancienne méthode de clustering sur deux maillages différents. Les images du centre montrent les singularités en violet et celles droite les différents clusters de directions stables choisies qui ne s'intersectent pas.

breuses manières de définir les clusters. Autorise-t-on des clusters à avoir des intersections non nulles ? Essaye-t-on de remplir entièrement le volume avec nos clusters ? Comment définir l'arrêt de la propagation du cylindre ? La FIGURE 24 illustre les résultats d'une ancienne méthode. Elle empêchait deux clusters de s'intersecter. Or il pouvait arriver qu'une direction se propage en premier dans une zone nécessaire à la propagation d'un second cluster qui ne pouvait alors plus se propager. C'est pourquoi la méthode retenue et présentée dans la section suivante permettra les intersections entre clusters.

1 Propagation des directions stables

On rappelle que l'on dispose d'un *framefield* initial que l'on note FF qui nous donne trois directions orthogonales en chaque tétraèdre. On cherche à construire des clusters de couples tétraèdre/direction $(c, \vec{z})$ autour des singularités où $\vec{z}$ est la direction stable choisie dans la croix FF_c de la cellule c. Pour cela, on parcourt les couples tétraèdre/direction en utilisant un algorithme de type Dijkstra en partant des singularités avec une distance radiale par rapport à la direction stable de propagation. La propagation s'arrête pour un cluster lorsqu'on rencontre un couple cellule/direction interdit. Ces couples $(c, \vec{z})$ interdits sont définis par les cellules c adjacentes aux arrêtes de singularités accompagnées d'une direction $\vec{z}$ non stable.

Dans l'algorithme imaginé nous avons besoin de tracer des *streamlines*, or nous avons un champ de croix composé de plusieurs vecteurs alors qu'auparavant nous tracions des *streamlines* dont la direction est donnée par un unique champ de vecteur. Nous devons donc donner une nouvelle définition. Une *streamline* commence dans une cellule c avec une direction $\vec{z} \in FF_c$. Ensuite la *streamline* est définie en suivant la direction $\vec{z}$ à l'intérieur de c puis lors d'une transition vers une nouvelle cellule c' la nouvelle direction utilisée est la direction $\vec{z}'$ la plus proche de $\vec{z}$ dans $FF_{c'}$, et ainsi de suite à chaque transition :

$$\vec{z}' = \underset{\vec{x} \in FF_{c'}}{\arg\max} \left| \vec{x}.\vec{z} \right|$$

Dans l'algorithme que l'on décrit, une *streamline* σ contient l'ensemble des couples cellule/direction traversés, et σ_c désigne la portion de la *streamline* qui traverse la cellule c.

L'initialisation de l'algorithme est donné par **Algorithme 1**. On y construit l'ensemble Z des couples $(c, \vec{z})$ qui serviront à stopper la propagation des clusters et on place dans une file de priorité les points de départs des clusters autour des singularités. Cette file contient des triplets composés d'une distance permettant d'établir l'ordre dans la file de priorité pour obtenir une isotropie de la propagation, d'un couple cellule/direction à visiter $(c, \vec{z})$ et d'un second couple prédécesseur $(c_0, \vec{z_0})$. Ce dernier permet de connaître le cluster qui souhaite se propager dans le couple à visiter. Cette information est utile pour soit fusionner les deux couples au sein du même cluster soit arrêter la propagation de celui-ci lorsque $(c, \vec{z})$ se trouve dans Z ou lorsque que les *streamlines* qui partent de $(c, \vec{z})$ ne terminent pas sur un bord (ces dernières peuvent par exemple finir par tourner presque indéfiniment dans un anneau et ne pas atteindre de bord en un temps limite).

Algorithme 1 : Initialisation de la propagation

```
1  Q ← ∅                                                          // file de priorité
2  Z ← ∅                                          // ensemble des couples (c,z) interdits
3  D tableau qui associe la valeur infinie INF à chaque couple (c,z)   // Distance radiale
4  pour chaque arrête singulière e faire
5      pour chaque tétraèdre c contenant e faire
6          Trouver la direction stable de FF_c par rapport à e : z
7          Trouver les direction instables de FF_c par rapport à e : x et y
8          Z ← Z ∪ {x, y}
9          d ← ‖ēc − (ēc . z) z‖      // distance radiale entre les barycentes de e, c
10         si d < D[(c, z)] alors
11             Q ← Q ∪ {(d, (c, z), (c, z))}
12             D[(c, z)] ← d
```

La suite (**Algorithme 2**) se comporte comme l'algorithme de plus court chemin de Dijkstra avec une notion d'adjacence un peu particulière. En effet un couple $(c, \vec{z})$ est adjacent à l'ensemble des cellules adjacentes à c accompagnées de la direction de leurs croix, la plus proche de

$\vec{z}$. Mais, on rajoute aussi à cette adjacence, l'ensemble des couples contenus dans les *streamlines* qui partent de $(c, \vec{z})$ et $(c, -\vec{z})$ afin d'obtenir cette structure cylindrique des clusters. Ensuite, le tableau *Stop* permet d'arrêter la propagation des couples d'un cluster qui seraient encore dans la file Q lorsqu'une condition d'arrêt à été rencontrée. Finalement c'est l'*Union Find C* qui, à la fin, donne les clusters après avoir fusionné tous les couples adjacents visités durant toute la propagation. A noter qu'on ne se retrouve pas nécessairement avec un cluster par singularité puisque les clusters de deux singularités de même direction stable côte à côte peuvent fusionner.

Algorithme 2 : Extension de la propagation

C *Union Find* sur l'ensemble des couples $(c, \vec{z})$
Vu tableau qui associe la valeur FAUX à chaque couple $(c, \vec{z})$
Stop tableau qui associe la valeur FAUX à chaque couple $(c, \vec{z})$
tant que Q *n'est pas vide* **faire**
    Récupérer le plus petit triplet $(\mathbf{d}, (\mathbf{c}, \vec{\mathbf{z}}), (\mathbf{c_0}, \vec{\mathbf{z_0}}))$ de **Q**
    si $\mathbf{d} = \mathbf{D}[(\mathbf{c}, \vec{\mathbf{z}})]$ *et non* $\mathbf{Stop}[\mathbf{C}.find(\mathbf{c_0}, \vec{\mathbf{z_0}})]$ **alors**
        σ_+, σ_- les *streamlines* partant de $(\mathbf{c}, \vec{\mathbf{z}})$ et $(\mathbf{c}, -\vec{\mathbf{z}})$
        si σ_+ *et* σ_- *terminent sur un bord et* $(\sigma_+ \cup \sigma_-) \cap \mathbf{Z} = \emptyset$ **alors**
            Fusionner $\mathbf{C}(\mathbf{c}, \vec{\mathbf{z}})$ et $\mathbf{C}(\mathbf{c_0}, \vec{\mathbf{z_0}})$
            $\mathbf{Vu}[(\mathbf{c}, \vec{\mathbf{z}})] \leftarrow$ VRAI
            pour chaque $\mathbf{c}'$ *adjacent à* **c faire**
                $\vec{\mathbf{z}}' \leftarrow \arg\max_{\vec{x} \in FF_{c'}} \left| \vec{x}.\vec{z} \right|$
                $\mathbf{d}' \leftarrow \mathbf{d} + \left\| \overrightarrow{\bar{\mathbf{c}}\bar{\mathbf{c}}'} - \left(\overrightarrow{\bar{\mathbf{c}}\bar{\mathbf{c}}'} . (\vec{\mathbf{z}}' + \vec{\mathbf{z}}) \right) (\vec{\mathbf{z}}' + \vec{\mathbf{z}}) \Big/ \left\| \vec{\mathbf{z}}' + \vec{\mathbf{z}} \right\|^2 \right\|$
                si $\mathbf{d}' < \mathbf{D}[(\mathbf{c}', \vec{\mathbf{z}}')]$ **alors**
                    $\mathbf{Q} \leftarrow \mathbf{Q} \cup \{(\mathbf{d}', (\mathbf{c}', \vec{\mathbf{z}}'), (\mathbf{c}, \vec{\mathbf{z}}))\}$
                    $\mathbf{D}[(\mathbf{c}', \vec{\mathbf{z}}')] \leftarrow \mathbf{d}'$
                si $\mathbf{Vu}[(\mathbf{c}', \vec{\mathbf{z}}')]$ **alors**
                    Fusionner $\mathbf{C}(\mathbf{c}, \vec{\mathbf{z}})$ et $\mathbf{C}(\mathbf{c}', \vec{\mathbf{z}}')$
            pour chaque $(\mathbf{c}', \vec{\mathbf{z}}') \in (\sigma_+ \cup \sigma_-)$ **faire**
                $\mathbf{d}' \leftarrow \mathbf{d} + \left\| \overrightarrow{\sigma_{c'}\mathbf{c}'} - \left(\overrightarrow{\sigma_{c'}\mathbf{c}'} . \vec{\mathbf{z}}' \right) \vec{\mathbf{z}}' \right\|$
                si $\mathbf{d}' < \mathbf{D}[(\mathbf{c}', \vec{\mathbf{z}}')]$ **alors**
                    $\mathbf{Q} \leftarrow \mathbf{Q} \cup \{(\mathbf{d}', (\mathbf{c}', \vec{\mathbf{z}}'), (\mathbf{c}, \vec{\mathbf{z}}))\}$
                    $\mathbf{D}[(\mathbf{c}', \vec{\mathbf{z}}')] \leftarrow \mathbf{d}'$
                si $\mathbf{Vu}[(\mathbf{c}', \vec{\mathbf{z}}')]$ **alors**
                    Fusionner $\mathbf{C}(\mathbf{c}, \vec{\mathbf{z}})$ et $\mathbf{C}(\mathbf{c}', \vec{\mathbf{z}}')$
        sinon
            $\mathbf{Stop}[\mathbf{C}.find(\mathbf{c_0}, \vec{\mathbf{z_0}})] \leftarrow$ VRAI

La Figure 25 montre des résultats obtenus avec cet algorithme. Les clusters sont suffisamment grands pour pouvoir correctement travailler sur les singularités et pas trop grand pour que l'algorithme puisse être rapide puisque le tracé des *streamlines* peut prendre du temps. Pour

ces maillages qui ont de l'ordre de la centaine de milliers de cellules, l'algorithme prend moins de 5sec alors qu'avec une propagation qui remplie le volume, ce temps pouvait être 10 fois plus long sur certains exemples.

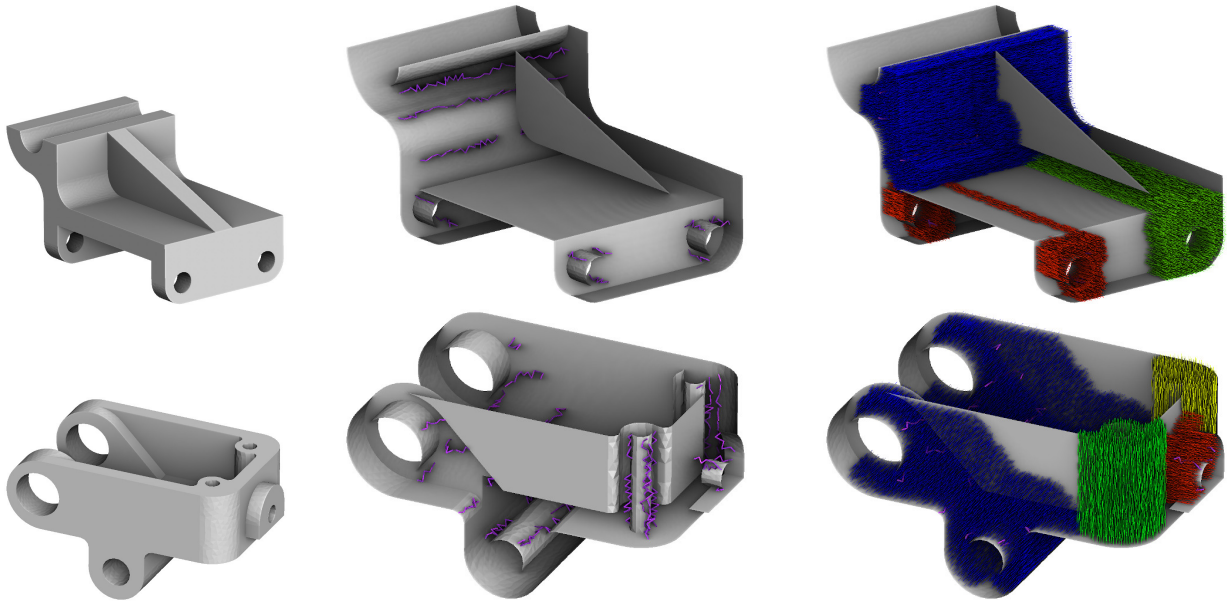


FIGURE 25 – Résultats de la propagation des directions stables. Les images du centre montrent les singularités en violet et celles droite les différents clusters de directions stables chacun représenté par une couleur. La direction des traits indique la direction stable.

2 Mise à jour locale du *framefield*

Une fois les clusters obtenus, on peut mettre à jour le *framefield* dans ces derniers en calculant un nouveau champ de croix avec l'idée de la 2.5D puisque le vecteur en second élément des couples qui constituent les clusters nous donne une direction stable correspondant à la direction de verticalité de la première partie. Les clusters pouvant s'intersecter, on fait cette mise à jour cluster après cluster. L'essentiel de ces calculs de *framefield* sont décrits dans la partie précédente. Toutefois, des changements ou nouveautés apparaissent sur plusieurs points donnés dans les paragraphes suivants.

Projection Ici le modèle n'a pas été déformé pour utiliser une projection verticale. La projection d'une cellule se fait donc en suivant une *streamline* partant de son barycentre jusqu'à rencontrer le bord du maillage. Pour se faire il faut effectuer un prétraitement des couples $(c, \vec{z})$ au sein d'un cluster C. Normalement la construction de l'ensemble des couples interdits Z de la section précédente empêche que le cluster C contienne deux couples différents avec la même cellule. On considère alors que C est un sous-ensemble de cellules c du maillage et que l'on dispose d'un champ vectoriel $\vec{z}$ sur ce dernier. Ensuite, la direction stable $\vec{z}$ étant donné à un signe près par l'algorithme de la section précédente, il faut redonner de la cohérence à tous ce champ. On propage alors la direction d'une cellule choisie arbitrairement au reste du cluster de sorte

que pour deux cellules c et c' adjacentes on ait $\vec{z}_c . \vec{z}_{c'} > 0$. La projection $p(c)$ de la cellule c est alors définie comme l'intersection entre le bord du maillage et la *streamline* partant de $\bar{c}$ et ayant pour vecteur vitesse $-\vec{z}$.

Murs On rappelle que seuls les triangles de murs doivent contraindre le champ de croix 2D calculé puisque que les sols et les toits on déjà la direction stable qui est normal au bord. Il y a d'ailleurs équivalence pour cette dernière propriété. On défini ainsi les murs comme les triangles $t \in (\partial M) \cap C$ dont la normale $\vec{n}_t$ vérifie $\left|\vec{n}_t . \vec{z}_c\right| < 0.8$, où c est la cellule à laquelle appartient le triangle t.

Base orthonormal Nous avions aussi eu besoin de construire des bases orthonormales en chaque cellule dans la première partie (sous-section [4.1]). Cette fois-ci nous disposons directement du troisième vecteur de la base $\vec{z}$ mais pas de $\vec{z}_0$. Le premier vecteur de la base $\vec{x}_0$ n'est donc pas choisi orthogonale à un certain $\vec{z}_0$ mais à un certain $\vec{z}_{c_0}$ où c_0 est une cellule choisie arbitrairement dans C. Puis pour chaque cellule c, les deux premiers vecteurs de la base sont définis par :

$$\vec{x}_c = (\vec{x}_0 - (\vec{x}_0 . \vec{z}_c)\vec{z}_c) \Big/ \left\|\vec{x}_0 - (\vec{x}_0 . \vec{z}_c)\vec{z}_c\right\| \qquad \text{et} \qquad \vec{y}_c = \vec{z}_c \wedge \vec{x}_c$$

Cette définition s'est avérée suffisante sur les maillages dont nous disposions mais on peut éventuellement imaginer que le champ $\vec{z}$ puisse varier beaucoup au sein du cluster pouvant possiblement atteindre une valeur proche de $\vec{x}_0$ et provoquer des discontinuités du champ $\vec{x}$ qui n'est pas défini et pas continue en $\vec{z}_c = \vec{x}_0$. Pour remédier à cela une solution envisageable est d'effectuer l'optimisation suivante sur ce champ $\vec{x}$:

$$\vec{x} = \arg\min_{\vec{v}} \sum_{c \in C} \left\|\nabla \vec{v}_c\right\|^2 \qquad \text{t.q. } \forall c \in C,\ \vec{v}_c . \vec{z}_c = 0 \text{ et } \|\vec{v}_c\| = 1$$

Cela permettrait d'avoir un champ lisse sans discontinuités et orthogonal à $\vec{z}$.

Bord interne au maillage Enfin le dernier point auquel il faut faire attention pour ce nouveau *framefield* 2.5D est la présence de triangle de bord de C qui sont des triangles internes au maillage, c'est à dire appartenant à $\partial C \setminus \partial M$. Il faut ajouter des contraintes pour ces triangles. Soit alors $c \in C$ et $c' \notin C$ des cellules adjacentes. La croix du *framefield* initial en c', $FF_{c'}$ peut être vue comme une direction stable $\vec{z}' = \arg\max_{\vec{x} \in FF_{c'}} \left|\vec{x} . \vec{z}_c\right|$ accompagnée d'une croix 2D, $(FF_{c'} \setminus \{\vec{z}', -\vec{z}'\})$. Soit $\vec{x}'$ un vecteur de cette croix 2D. Ce vecteur peut être décrit par l'angle suivant dans la base de c :

$$\phi_{c'} = \operatorname{atan2}(\vec{x}' . \vec{y}_c,\ \vec{x}' . \vec{x}_c)$$

Cet angle nous permet de définir la représentation vectorielle $(\cos(4\phi_{c'}), \sin(4\phi_{c'}))$ (Figure 20) de cette croix. Puis tout comme on minimise le gradient au carré de cette représentation vectorielle à l'intérieur du maillage on fait de même ici pour éviter toute coupure entre le cluster et le reste du maillage en ajoutant le terme suivant :

$$\sum_{\substack{c \sim c' \\ c \in C,\ c' \notin C}} \frac{1}{l_{cc'}} \left((a_c - \cos(4\phi_{c'}))^2 + (b_c - \sin(4\phi_{c'}))^2\right)$$

aux fonctionnelles que l'on minimise à l'initialisation du *framefield* (20) ainsi qu'à chaque itération (21).

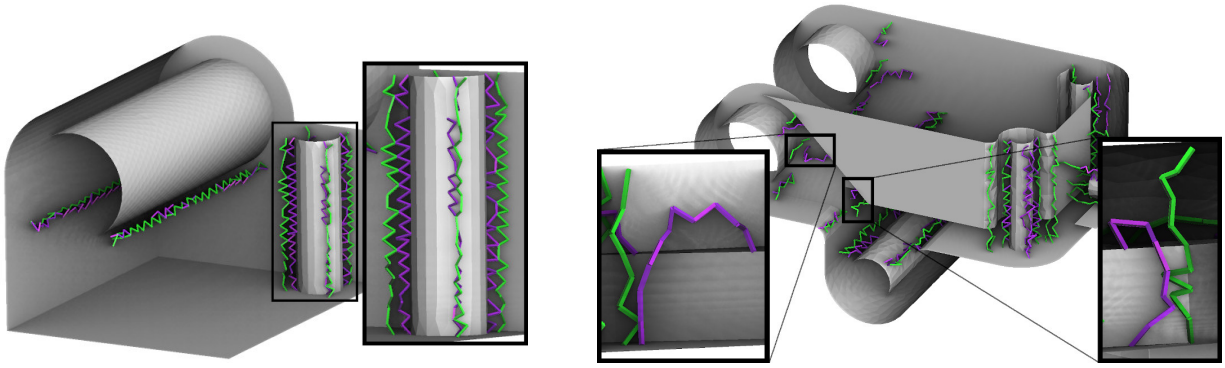


FIGURE 26 – Résultats de la mise à jour du *framefield*. Sur ces deux maillages les singularités du champ de départ sont données en violet tandis que les singularités en sortie de notre algorithme sont données en vert.

Résultats Comme espéré, ces mise à jours locales du *framefield* permettent de rendre valide certaines singularités invalides du *framefield* d'entrée. Cette correction est visible sur le maillage de droite de la FIGURE 26. De plus sur l'image de gauche de la même figure on peut voir que des singularités sont éloignées du bord grâce à notre algorithme. Ce phénomène apparaît sur beaucoup de maillages. C'est un résultat très intéressant car les singularités qui rentrent dans les murs ne sont pas appréciées pour les simulations numériques. Et c'est exactement ce qui se produisait sur le champ initial du maillage de gauche. On peut voir par exemple en haut à gauche du tube sur lequel le zoom est effectué que la singularité violette disparaît en rentrant dans le mur et réapparaît une arrête plus loin, ce qui est ensuite corrigé. Enfin, ces résultats sont très satisfaisant, car ils sont robustes et permettent de rende valide des *framefield* qui ne l'étaient pas afin de permettre la génération de maillages hexaédriques qui ne pouvait pas être obtenus à causes des anciennes singularités invalides. De plus les futurs travaux de recherches à propos des *framefield* 2D pourront directement impacter la génération de champ de croix 3D via l'algorithme présenté.

Conclusion

Bien qu'ayant essayé de construire un algorithme permettant de générer depuis zéro un *framefield* 3D en faisant une extrusion d'un champ de croix 2D, ce stage aura finalement permis de construire un algorithme qui permet de corriger un premier *framefield* en utilisant la même idée de la 2.5D. Notre objectif initial s'est confronté à la difficulté de remplir un maillage avec un champ de direction stable correct. Nous avons tout de même réussi à obtenir des résultats satisfaisants sur des maillages très simples possédant une seule direction stable constante. Mais, la faible quantité de maillages sur laquelle cette méthode fonctionnait fait que les résultats n'étaient pas si intéressants que ça.

De longues réflexions nous ont amené à la conclusion que remplir un modèle avec des directions stables n'était pas nécessairement beaucoup plus simple que de générer un *framefield* entier. Ce dernier permettrait de récupérer des directions stables parmi les trois directions données par chaque croix du champ. A ce moment, le stage a pris une autre direction. La 2.5D peut être utilisée pour rendre valide un graphe de singularités d'un *framefield* préalablement calculé via des méthodes qui ne garantissent pas la validé des singularités. Les résultats obtenus sont satisfaisants d'autant plus qu'ils sont nécessaires pour pouvoir garantir la bon fonctionnement de l'intégration du champ de croix qui sert à produire le maillage hexaédrique voulu. Malheureusement les corrections que notre algorithme apporte aux *framefields* ne sont pas suffisantes pour obtenir un maillage hexaédrique correct. Certains maillages contiennent ce que les membres de l'équipe PIXEL appellent des "tremplins". Lors de l'étape de l'intégration du champ, des valeurs entières sont imposées à une coordonnée de la paramétrisation pour chaque singularité et chaque face du bord. Il peut arriver qu'un morceau en forme de tremplin sorte d'une face avec un angle assez faible pour que le haut du tremplin ne soit pas considéré orthogonal à la face. La face et le haut du tremplin se voient alors associer la même valeur entière selon l'une des coordonnées de la paramétrisation, ce qui conduit à l'apparition de cellules toutes allongées comme on peut le voir Figure 28. Notre méthode ne peut pas empêcher ce genre de phénomènes d'arriver.

Toutefois, l'équipe PIXEL commence à s'intéresser à la génération de *framefield* non orthogonaux, notamment en 2D. C'est à dire à des champs de croix pour lesquels on n'impose plus de conditions d'orthogonalité entre les différentes branches de chaque croix. Cela permettrait d'avoir une direction des croix orthogonale au haut du tremplin et une seconde direction orthogonale à la face sur laquelle repose le tremplin (voir Figure 27). La méthode proposée dans la seconde partie de ce stage permettrait de ne s'intéresser qu'au cas 2D, puisque celui ci pourrait être directement transposé au cas 3D via la 2.5D.

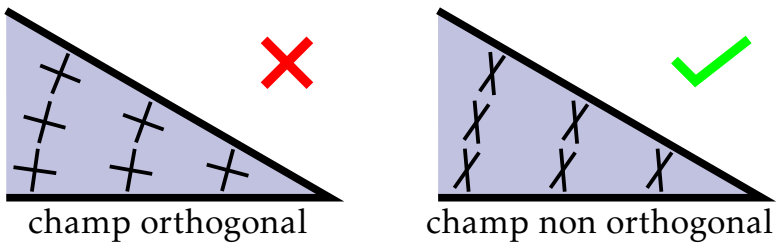


Figure 27 – Champs de croix orthogonaux et non orthogonaux sur un tremplin.

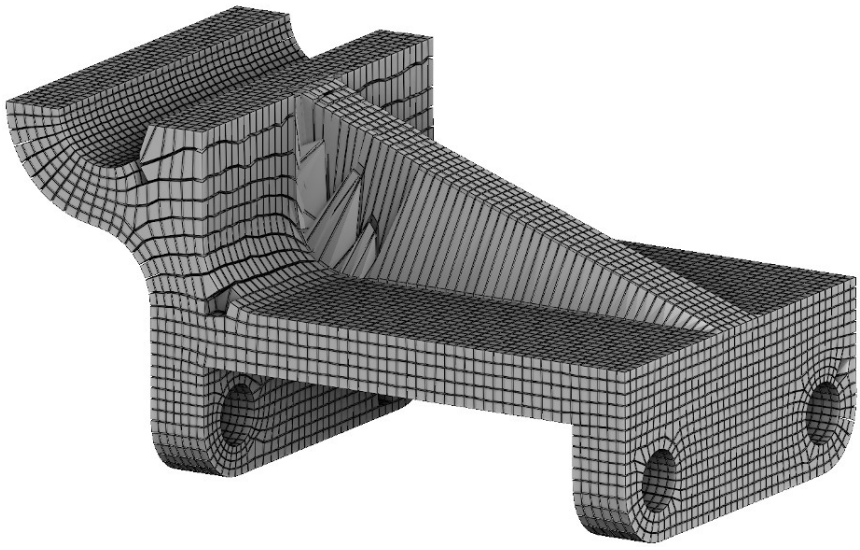


Figure 28 – Exemple de tremplin.

Références

[Cra19] Keenan Crane. The n-dimensional cotangent formula. 2019.

[CWW17] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. The heat method for distance computation. *Commun. ACM*, 60(11) :90–99, October 2017.

[FSM+17] Harold J. Fogg, Liang Sun, Jonathan E. Makem, Cecil G. Armstrong, and Trevor T. Robinson. A simple formula for quad mesh singularities. *Procedia Engineering*, 20 :14–26, October 2017.

[GSZ11] J. Gregson, A. Sheffer, and E. Zhang. All-hex mesh generation via volumetric polycube deformation. *Computer Graphics Forum (Special Issue of Symposium on Geometry Processing 2011)*, 30(5) :to appear, 2011.

[KNP07] Felix Kälberer, Matthias Nieser, and Konrad Polthier. Quadcover - surface parameterization using branched coverings. *Comput. Graph. Forum*, 26(3) :375–384, 2007.

[LBK16] Max Lyon, David Bommes, and Leif Kobbelt. Hexex : robust hexahedral mesh extraction. *ACM Trans. Graph.*, 35(4) :123 :1–123 :11, 2016.

[LG15] Zohar Levi and Craig Gotsman. Smooth rotation enhanced as-rigid-as-possible mesh animation. *IEEE Trans. Vis. Comput. Graph.*, 21(2) :264–277, 2015.

[LLX+12] Yufei Li, Yang Liu, Weiwei Xu, Wenping Wang, and Baining Guo. All-hex meshing using singularity-restricted field. *ACM Trans. Graph.*, 31(6) :177 :1–177 :11, 2012.

[NRP11] Matthias Nieser, Ulrich Reitebuch, and Konrad Polthier. Cubecover- parameterization of 3d volumes. *Comput. Graph. Forum*, 30(5) :1397–1406, 2011.

[Owe] Steven J. Owen. An introduction to automatic mesh generation algorithms - part ii.

[RS15] Nicolas Ray and Dmitry Sokolov. On smooth 3d frame field design. *CoRR*, abs/1507.03351, 2015.

[SA07] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In Alexander G. Belyaev and Michael Garland, editors, *Proceedings of the Fifth Eurographics Symposium on Geometry Processing, Barcelona, Spain, July 4-6, 2007*, volume 257 of *ACM International Conference Proceeding Series*, pages 109–116. Eurographics Association, 2007.

[SCV14] Justin Solomon, Keenan Crane, and Etienne Vouga. Laplace-beltrami : The swiss army knife of geometry processing. Symposium on Geometry Processing, July 7 2014.