AMS314 Rapport du Projet

Partie 1

Boqiao HUANG mars 2025

Organization des ouputs

Les outputs se trouvent dans le dossier *ouput*. Donc il est conseillé de lancer le code dans le dossier *ouput* pour que les fichiers puissent être mis à jour automatiquement.

- `connex.solb` est la solution connexe pour la partie III.
- `quality1.solb` et `quality2.solb` sont les solutions de qualité avec critère Q_1 et Q_2.
- `TriVoi_Hash.txt` et `TriVoi_Q2.txt` sont les listes des voisinage que peuvent valider mon développement.

I. Qualité de maillage

D'abord, j'ai calculé les valeurs de α_1 et α_2 :

$$\alpha_1 = \frac{1}{4\sqrt{3}}, \qquad \alpha_2 = \frac{\sqrt{3}}{6}$$

Ensuite, en utilisant les formules dans le cours, j'ai implémenté les fonctions pour calculer l'aire K et le rayon du cercle inscrit ρ. La qualité de maillage de `carre_4h.mesh` évaluée par Q_1 et Q_2 est présenté dans Fig. 1 et Fig. 2.

Il est constaté que les qualités des triangles sont très proches de 1. En effet, la plupart des triangles sont très proches du triangle équilatéral. En plus, la qualité évaluée par Q_1 est plus proche de 1 que celle évaluée par Q_2. En effet, Q_1 moyenne l'effet de chaque arête, alors que pour Q_2 la qualité augmente tant que les longeurs des trois arêtes d'un triangle sont déséquilibrées. Donc, c'est le critère Q_2 qui est plus fidèle pour déterminer à quel point qu'un triangle est proche du triangle équilatéral.

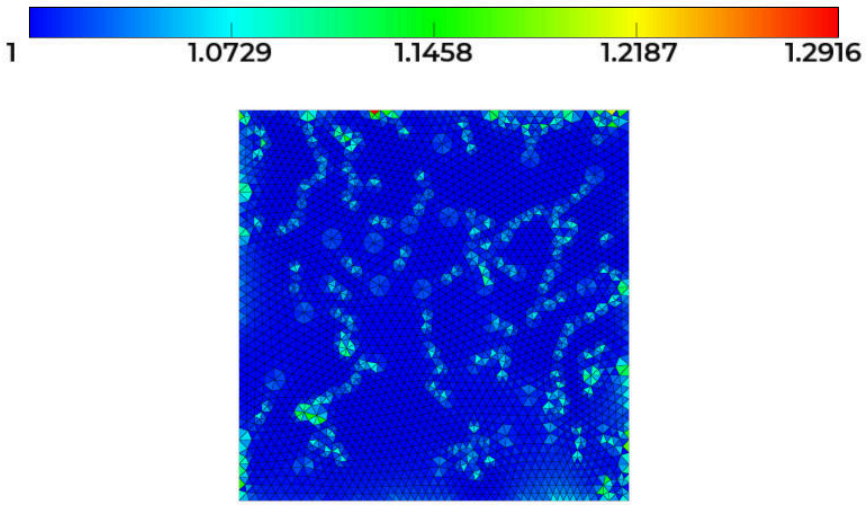

Fig. 1: La qualité de maillage évaluée par Q_1

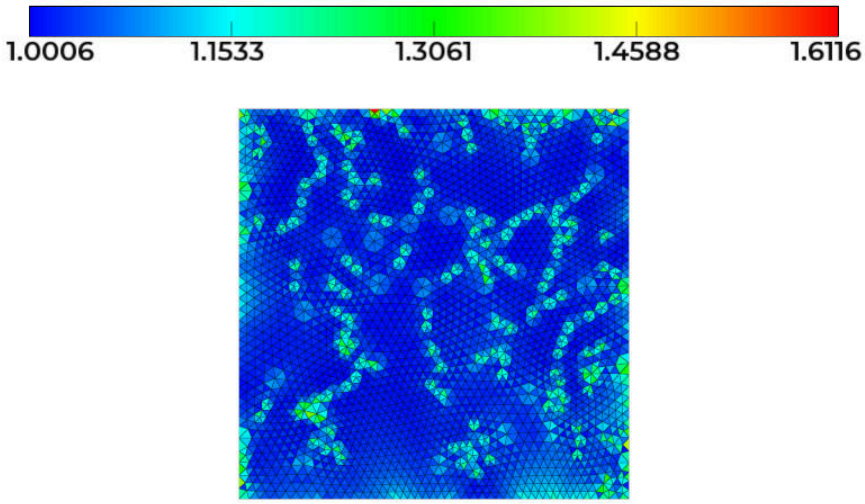

Fig. 2: La qualité de maillage évaluée par Q_2

II. Comparaison entre l'approche naïve et l'approche hashage

J'ai réalisé la table de hashage en suivant la démarche décrite dans l'énoncé. Suite à la réalisation, j'ai comparé le temps de lecture de différents maillages par l'approche naïve et l'approche basée sur la table de hachage, dont les résultats sont dans Tableau 1. Il faut noter que j'ai utilisé la clé $i + j$ pour ces essais. Le comptage de collisions reste activé pendant les essais, car il a peu d'effet sur le temps de calcul.

	`carre_4h.mesh`	`carre_2h.mesh`	`carre_h.mesh`	`carre_05h.mesh`
naïve (seconde)	0.152	2.51	44.5	737
hachage (seconde)	0.00178	0.0130	0.101	0.728
Nbr de vertices	2625	10663	43758	178746
Nbr de triangles	5084	20938	86742	355946

Tableau 1: Temps de lecture avec l'approche naïve et hachage

II.1. Complexité temporelle

Théoriquement, l'approche naïve a une complexité de $O(N^2)$ et l'approche hachage a une complexité de $O(N)$, avec N le nombre de triangles. Fig. 3 illustre la relation logarithmique entre le nombre de triangles et le temps de calcul pour les maillage testés dans Tableau 1. Il est observé que

1) pour l'approche naïve, la pente de la régression linéaire est 2, ce qui signifie que la complexité est bien quadratique.
2) pour l'approche hachage, la complexité mesurée est entre linéaire et quadratique, différente que le résultat théorique. Une explication possible est que les collisions augmente la complexité.

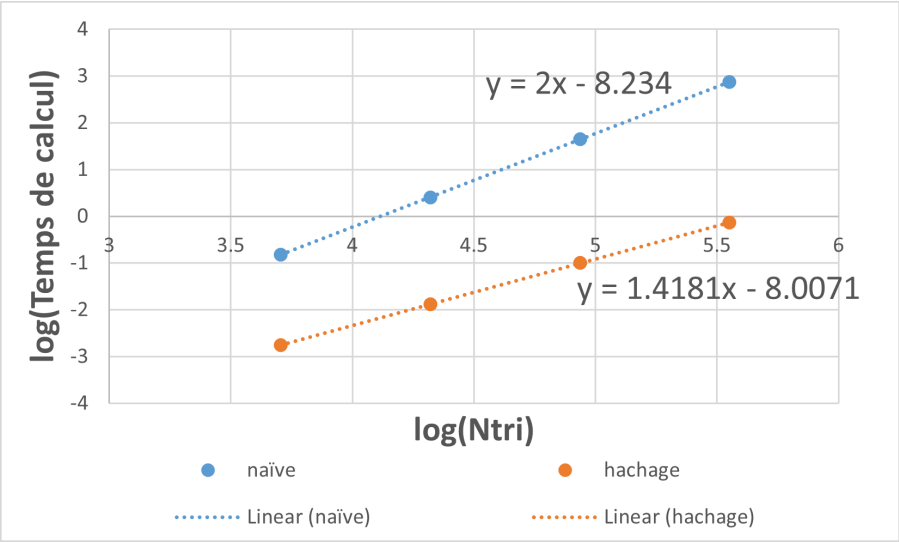

Fig. 3: Relation logarithmique entre le nombre de triangles et le temps de calcul

II.2. Nombre d'arêtes frontières

Cette fonctionalité est réalisée par la fonction `compute_NbrEdgBoudry` dans le code source. L'idée est de parcourir chaque arête de chaque triangle et de vérifier si son `TriVol[iTri][iEdg]` est 0 : si oui, c'est une arête frontière. Tableau 2 présente le nombre d'arêtes frontières pour différents maillages.

	`naca0012.mesh`	`ls89.mesh`	`hlcrm.mesh`
Nbr d'arêtes frontières	492	380	690

Tableau 2: Nombre d'arêtes frontières

II.3. Nombre de collisions

Il faut noter tout d'abord la convention de collision que j'ai prise :

Si n $(n > 1)$ éléments partagent la même clé,
alors on dit qu'il y a $(n-1)$ collisions pour cette clé.

Cette fonctionalité est réalisée par la fonction `collisions` dans le code source. L'idée est que si un élément de `Head` n'est pas 0, alors il y a un ou plusieurs arêtes dans cette clé. Ensuite, on va dans cette clé et compter le nombre de collisions en vérifiant si `LstObj[id][4]` d'une arête est 0 : si non, il y a une autre arête qui le suit, donc une collision. Cette démarche permet d'obtenir le nombre de collisions maximum, le nombre de collisions total et le nombre de clés où il y a des collisions. Ensuite, le nombre de collisions moyen se calcul par (le nombre de collisions total) / (le nombre de clé où il y a des collisions).

En plus des clés *min* et *sum*, j'ai proposé une autre clé qui est définie par

$$\text{key} = \left\lfloor \left(1 + \frac{\min(\text{iVer1}, \text{iVer2})}{\max(\text{iVer1}, \text{iVer2})}\right) \times \text{facteur} \right\rfloor$$

Normalement, pour garantir suffisamment de variété, le facteur doit être de grandeur suivant

$$\frac{1}{\text{facteur}} \approx \min_{\substack{\forall \text{iVer1} \\ \forall \text{iVer2}}} \left(\frac{\min(\text{iVer1}, \text{iVer2})}{\max(\text{iVer1}, \text{iVer2})}\right) > \frac{1}{\text{iVer}_{\max}}$$

Dans le code, j'ai pris 10^7 comme facteur. La clé est borné entre $1 \times \text{facteur}$ et $2 \times \text{facteur}$ pour n'importe quel maillage. Cela nous permet d'appliquer cette clé sans changer le type `int` de la variable `key` même lorsque l'on a un très grand nombre de sommets (il suffit d'augmenter le facteur), ce qui est un avantage par rapport la clé *multiplication* (iVer1$\times$ iVer2), *sum* et *min*.

En plus, cette clé est censée d'avoir moins de collisions par rapport à la clé *sum*, car le résultat de la division est plus aléatoire.

Les résultats du nombre de collisions pour différents maillages sont présentés dans Tableau 3. On peut choisir la clé en changeant la variable `keymode` dans `main_mesh.c`.

		`naca0012.mesh`	`ls89.mesh`	`hlcrm.mesh`
clé $i+j$	Nbr de collisions max	14	9	7
	Nbr de collisions moyen	0.94	1.33	0.86
clé $\min(i,j)$	Nbr de collisions max	5	7	7
	Nbr de collisions moyen	1.89	2.12	2.00
clé division	Nbr de collisions max	6	6	7
	Nbr de collisions moyen	0.344	0.332	0.364

Tableau 3: Nombre de collisions

Il est observé que

1) pour la clé $\min(i,j)$, le nombre de collisions maximum est en général plus faible par rapport à la clé *sum*. En effet, ce nombre est majoré par le nombre de triangles qui partagent un même sommet. Cela fait que le nombre de collisions maximum dépassera rarement 10 si chaque triangle est de "bonne qualité".
2) pour la clé $\min(i,j)$, le nombre de collisions moyen est en général plus grand. En effet, comme chaque sommet est partagé par plusieurs arêtes, c'est plus probable que deux arêtes ont la même valeur de $\min(i,j)$.
3) la clé *division* a le nombre de collisions moyen le plus faible, ce qui est cohérent avec mon analyse précédante. Même si elle amène plus de calcul flottant, la clé *division* reste compétitive au niveau du temps de calcul. Tableau 4 montre les temps de lecture de l'algorithme hachage utilisant la clé *sum* et *division* avec le facteur 10^7.

	`carre_4h.mesh`	`carre_2h.mesh`	`carre_h.mesh`	`carre_05h.mesh`
clé sum (seconde)	0.00220	0.0141	0.0909	0.716
clé division (seconde)	0.00321	0.0159	0.0995	0.778

Tableau 4: Temps de lecture avec clé *sum* et *division*

III. Composantes connexes

Cette partie est réalisée dans la fonction `find_connex_components`. L'idée est la même que celle décrite dans l'énoncé. Finalement, j'ai retrouvé les composantes connexes du `squarecircle.mesh` dans Fig. 4. Le nombre de chaque composante connexe est présenté dans Tableau 5.

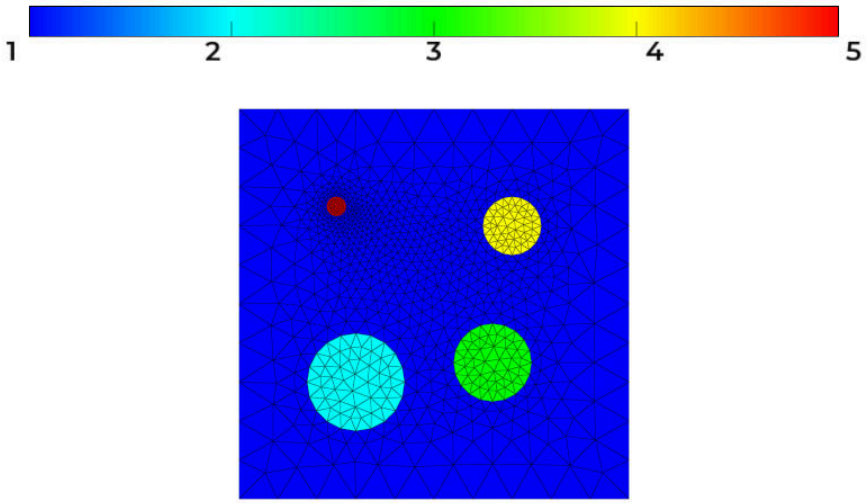

Fig. 4: Composantes connexes du squarecircle.mesh

	composante 1	2	3	4	5	total
Nbr de composante connexe	1316	88	88	88	88	1668

Tableau 5: Nombre de composantes connexes