Analyse spectrale des surfaces

Travaux pratiques

1 Objectifs du TP

Lors de ce TP, vous allez manipuler les objets mathématiques présentés dans le cours d'analyse spectrale :

1. matrice Laplacienne
2. diagonalisation *efficace* de l'opérateur Laplacien discrétisé
3. utilisation de la diagonalisation (vecteurs propres, valeurs propres) et liens avec l'analyse spectrale et les nombreuses notions de distances sur les variétés

2 Base de code

Au début du TP, vous recevrez une archive zip contenant un viewer de base, ainsi que des classes basées sur la librairie `Eigen` et `Spectra`.

L'objectif de ce TP n'est pas de comprendre les outils d'optimisation numérique, ni que vous passiez du temps à appréhender les librairies, mais que vous vous appropriez des notions géométriques sur les surfaces triangulées liées à l'analyse spectrale et l'opérateur Laplacien.

Pour compiler avec cmake, ouvrez un terminal dans le répertoire contenant CMakeLists.txt, et tapez :

— mkdir build
— cd build
— cmake -DCMAKE_BUILD_TYPE=Release ..
— cd ..
— cmake –build build

Important : si vous ne compilez pas en mode Release, votre programme prendra plusieurs minutes pour calculer les vecteurs propres du Laplacien sur les modèles les plus gros.

Pour éxécuter votre programme :

./main

3 Opérateur Laplacien

On demande de coder dans cet exercice la matrice Laplacienne *intégrée* sur les cellules associées aux sommets, ainsi que la matrice de masses associées aux sommets.

Pour rappel, la matrice Laplacienne (point-wise) est donnée par :

$$\begin{cases} \Delta(i,j) & = \frac{1}{|v_i|}\frac{cot(\alpha_{ij})+cot(\beta_{ij})}{2} \\ \Delta(i,i) & = -\sum_{j\neq i}\Delta(i,j) \end{cases} \tag{1}$$

Cet opérateur n'est pas symétrique. On ne peut donc pas le diagonaliser de manière triviale.

À la place, on considère la matrice Laplacienne *intégrée* L :

$$\begin{cases} L(i,j) & = \frac{cot(\alpha_{ij})+cot(\beta_{ij})}{2} \\ L(i,i) & = -\sum_{j\neq i} L(i,j) \end{cases} \tag{2}$$

et la matrice diagonale des masses associées aux sommets :

$$M(i,i) = |v_i| \tag{3}$$

On note immédiatement que $\Delta = M^{-1}.L$. On parle de Laplacien intégré car il est obtenu en considérant que le Laplacien est constant sur chaque cellule associée aux sommets, et le Laplacien intégré correspond alors bien à l'intégration du Laplacien sur la cellule associée aux sommets (d'aires $|v_i|$).

3.1 Diagonalisation

On parlera des valeurs propres λ_i et vecteurs propres ϕ_i du Laplacien comme étant les quantitées vérifiant le problème spectral généralisé suivant :

$$L.\phi_i = \lambda_i M.\phi_i \tag{4}$$

On *normalisera* les vecteurs propres afin d'obtenir :

$$\phi_i^T \cdot M \cdot \phi_j = \delta_i^j \tag{5}$$

Notez que ce produit discrétise le produit scalaire entre les fonctions $\phi_i(x)$ et $\phi_j(x)$ sur la surface : $\phi_i^T \cdot M \cdot \phi_j = \sum_v |v| \phi_i(v)\phi_j(v) \simeq \int_x \phi_i(x)\phi_j(x)ds(x)$.

On notera que les valeurs propres du Laplacien sont *négatives* (et nulle pour la première puisque le Laplacien d'une fonction constante vaut bien 0 partout). Cela a une conséquence pratique (pas particulièrement dérangeante), car la plupart des librairies informatiques existantes permettant l'extraction de vecteurs propres de matrices demandent que les valeurs propres soient positives. Cela explique que dans le code fourni, on effectue la diagonalisation du problème généralisé inverse, puis que l'on corrige les valeurs propres obtenues.

3.2 Exercice

1. Complétez le code de la fonction `computeLaplacianMatrix()`

2. Complétez le code de la fonction `computeMassMatrix()`, on se contentera d'attribuer à un sommet le tiers de l'aire des triangles qui y sont adjacents

3. Dans la fonction `draw()`, utilisez `draw_exercise_1()` afin de visualiser les vecteurs propres du Laplacien (flèches gauche et droite pour changer de vecteur propre).

3.3 Exercice : commute time distance

La "commute time distance" entre x_i et x_j décrit la distance (en termes d'espérance statistique) pour un marcheur aléatoire pour partir du point x_i, passer par x_j, et revenir à x_i.

De manière assez remarquable, cette distance s'exprime de manière analytique en fonction du spectre du Laplacien :

$$commuteTimeDist(x, y) = \sum_{i>0} \frac{(\phi_i(x) - \phi_i(y))^2}{|\lambda_i|} \tag{6}$$

Complétez le code de la fonction `computeDiffusionDistance(...)` et exécutez le code sur l'exemple du maillage `camel`.
Vous devriez obtenir le résultat suivant :

3.4 Exercice : biharmonic distance

La distance biharmonique est une autre distance intrinsèque liée à l'opérateur Laplacien, qui ressemble étrangement à la commute time distance. Elle s'exprime sous la forme suivante :

$$biharmonicDist(x, y) = \sum_{i>0} \frac{(\phi_i(x) - \phi_i(y))^2}{\lambda_i^2} \tag{7}$$

Complétez le code de la fonction `computeBiharmonicDistance(...)` et exécutez le code sur l'exemple du maillage `camel`.
Vous devriez obtenir le résultat suivant :
Comparez les deux distances obtenues.

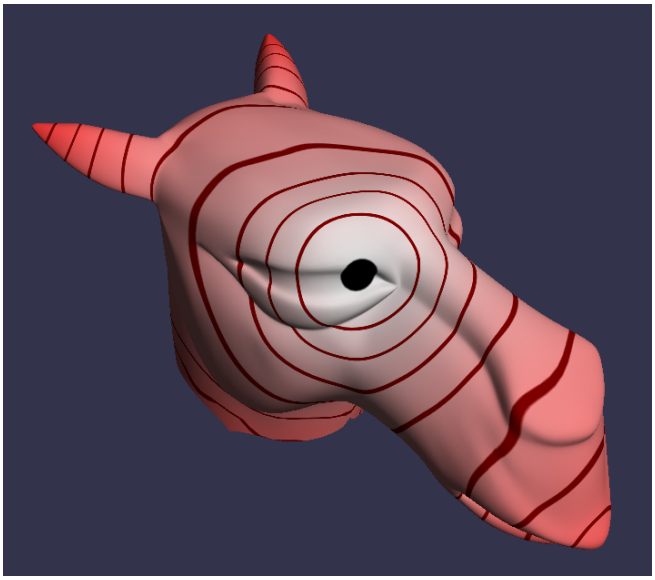

FIGURE 1 – Visualisation de la "commute time distance".

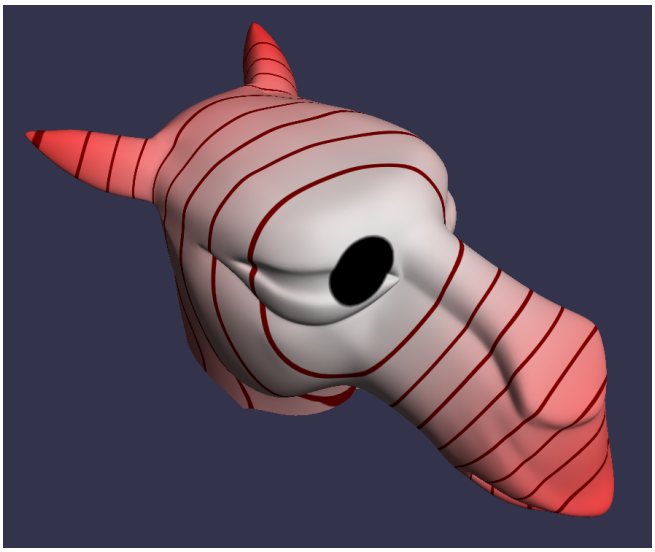

FIGURE 2 – Visualisation de la distance biharmonique.

3.5 Exercice : distance de diffusion

La distance de diffusion est *encore une autre* distance intrinsèque liée à l'opérateur Laplacien. Elle dépend d'un paramètre de temps t, et s'exprime sous la forme suivante :

$$diffusionDist_t(x,y) = \sum_{i>0} (\phi_i(x) - \phi_i(y))^2 e^{2\lambda_i t} \tag{8}$$

La distance de diffusion estime en moyenne la distance parcourue pour un marcheur aléatoire pour partir du point x_i, passer par x_j, et revenir à x_i, et cela sur l'ensemble des marches aléatoires réalisées en un temps donné t.

Notez le lien entre cette distance et la commute time distance.

Complétez le code de la fonction `computeBiharmonicDistance(...)` et exécutez le code sur l'exemple du maillage `camel` pour des valeurs de t variées entre 0.05 et 2.

Vous devriez obtenir le résultat suivant :

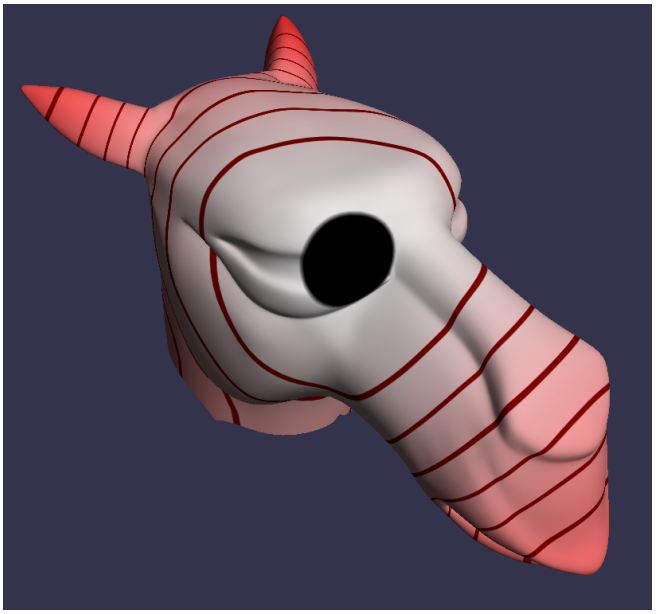

FIGURE 3 – Visualisation de la distance de diffusion pour t = 0.1.

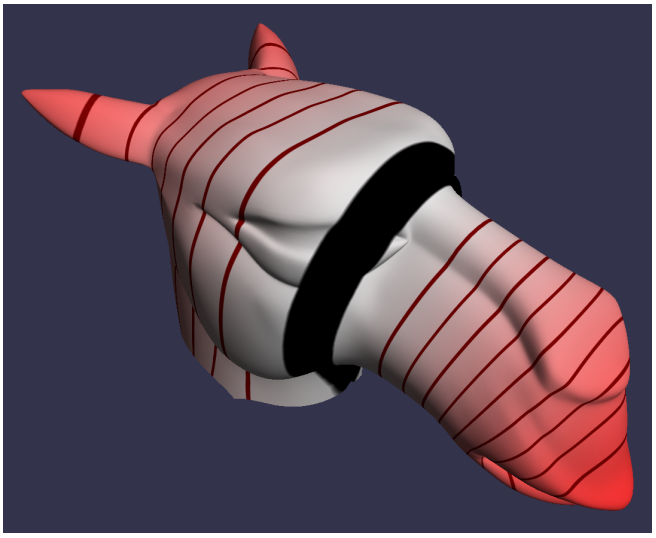

FIGURE 4 – Visualisation de la distance de diffusion pour t = 0.5.

Comment expliquez vous intuitivement le comportement de cette distance en fonction de t ?
Comparez la structure des distances obtenues avec la structure des vecteurs propres du Laplacien.

3.6 Exercice : noyau de la chaleur

Le noyau de la chaleur $k_t(x, y)$ décrivant la portion de chaleur transmise de x à y après un temps t se calcule de la manière suivante :

$$k_t(x, y) = \sum_{i \geq 0} \phi_i(x)\phi_i(y)e^{\lambda_i t} \tag{9}$$

Ce noyau apparait lorsque l'on écrit l'équation de la chaleur sur la base des vecteurs propres du Laplacien (voir cours, et démonstration au tableau)
Implémentez le noyau $k_t(x, x)$, qui est utilisé de manière courante comme descripteur multi-résolution des surfaces.

Vous devriez obtenir le résultat suivant :

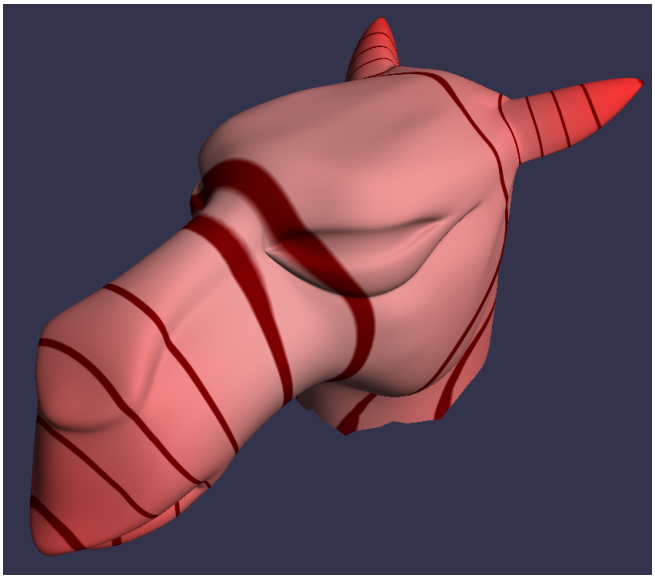

FIGURE 5 – Visualisation du noyau de la chaleur pour t = 0.2.

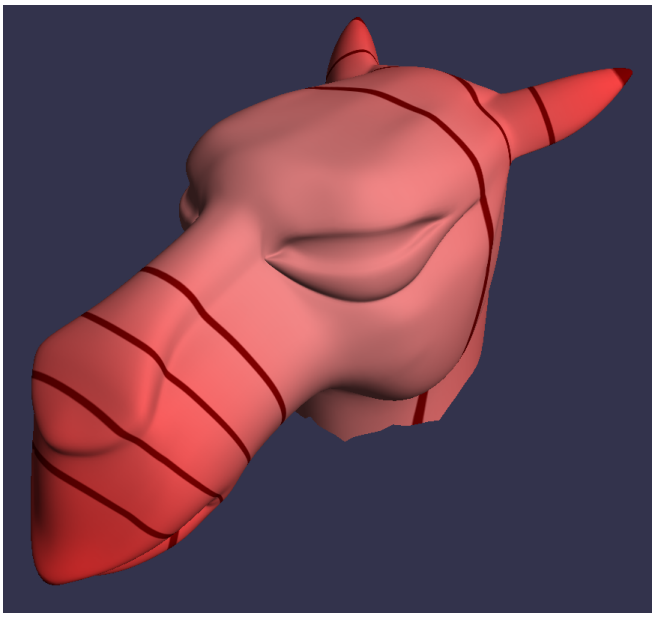

FIGURE 6 – Visualisation du noyau de la chaleur pour t = 0.35.

3.7 Exercice : distance géodésique et diffusion de la chaleur

Comme vu en cours, la chaleur se diffuse aux "petits temps" dans les directions géodésiques. On peut donc regarder dans quelle direction se propage la chaleur, et en déduire que le gradient de la distance géodésique est aligné sur le gradient obtenu.

Une distance géodésique approximée peut donc être obtenue avec l'algorithme suivant :

1. Résoudre l'équation de la chaleur $\partial_t U_t = \Delta U_t$, de manière approximée :
   $U_t - U_0 = t.\Delta U_t$

$M.(U_t - U_0) = t.M.\Delta U_t$ (intégrer l'équation sur les cellules des sommets)
$(M - t.L).U_t = M.U_0$ (c'est donc le système linéaire que l'on résoud)

2. calculer les gradients de la solution obtenue :
   $g = G.U_t$
3. inverser et normaliser ces gradients
   $g = -g/\|g\|$
4. Obtenir une approximation de la distance géodésique d en résolvant le système de Poisson suivant :
   $G.d = g$

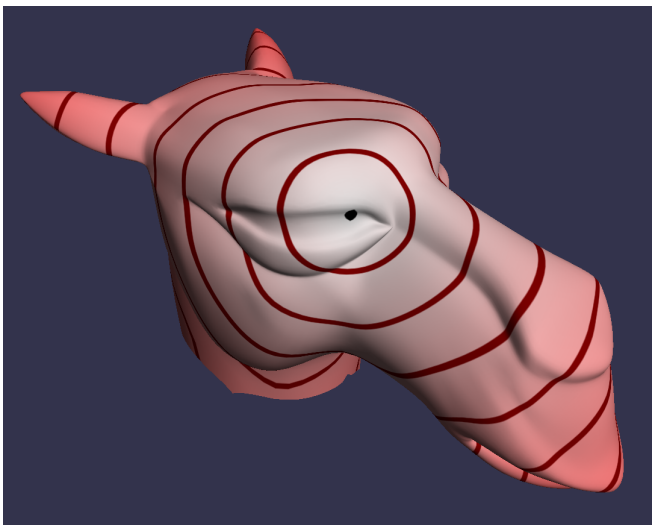

FIGURE 7 – Visualisation de la distance géodésique approximée à l'aide de l'équation de la chaleur.