

---

# Traitements géométriques de points 3D

## Travaux pratiques : Surfaces MLS

Au début de ce TP/TD, vous recevrez une archive zip contenant une base de code. Ce code permet d'afficher un *pointset* à l'aide d'OpenGL. La visualisation est extrêmement basique : le rendu n'est pas l'objectif du TP.

1. Nous commencerons par l'analyser ensemble pour vous familiariser avec.
2. Vous devez faire évoluer ce code au fur et à mesure du TP, pour répondre aux questions.
3. **Vous devrez m'envoyer avant le 3 juin à 23h59 un zip contenant votre fichier tp.cpp (qui doit contenir tout votre code), ainsi que votre rapport contenant une description très brève des questions que vous avez traitées et des captures d'écran présentant vos résultats.**
4. Faites un effort pour que le code soit lisible!!!!
5. Vous pouvez me poser des questions (jthierry@telecom-paristech.fr), j'y répondrai. Je ne toucherai pas à votre code par contre.
6. Vous devez travailler individuellement sur votre TP. Je vous fais confiance sur ce point.

## 1 Base de code

Téléchargez l'archive.

Pour le compiler, ouvrez une console, placez vous dans le répertoire principal, et tapez :

```
1 make
```

Pour l'exécuter, tapez :

```
./tp
```

### 1.1 Principaux éléments fournis

1. un viewer simple sous `glut`
2. une classe de kd-tree utilisant la librairie open source `ann`
3. une classe point 3D (`Vec3`, dans `Vec3.h`).

## 2 Modèles

Le répertoire `./pointsets` contient des fichiers binaires encodant des pointsets, sous la forme  $x, y, z, nx, ny, nz, \dots$ . Chaque modèle apparaît sous plusieurs formes :

1. `model.pn`
2. `model.subsampled.np` : est une version décimée de `model.pn`
3. `model.subsampled.extreme.np` : est une version **extrêmement** décimée de `model.pn` (contient entre 10 et 15% des points seulement)

Ces modèles serviront à tester vos algorithmes.

## 3 Exercice : HPSS

Implémentez une fonction :

```
5 void HPSS( Vec3 inputPoint ,  
            Vec3 & outputPoint , Vec3 & outputNormal ,  
            std::vector<Vec3> const & positions , std::vector<Vec3> const & normals , BasicANNkdTree const & kdtree ,  
            int kernel.type , float radius , unsigned int nbIterations = 10 , unsigned int knn = 20 ) {  
    ...  
}
```

- 
1. `Vec3 inputPoint` est votre point d'entrée.
  2. `Vec3 & outputPoint` , `Vec3 & outputNormal` sont la position et la normale obtenues en projetant `inputPoint` sur le `pointset`.
  3. `std::vector<Vec3> const & positions` , `std::vector<Vec3> const & normals` décrivent les positions et les normales du `pointset` à partir duquel vous voulez définir votre surface.
  4. `BasicANNkdTree const & kdtree` est le kd-tree que vous devez construire en preprocess à partir de `positions` (voir `int main()` pour un exemple d'utilisation).
  5. `int kernel_type` vous permettra de spécifier le type de noyau (singulier pour obtenir une surface *interpolante*, Gaussien ou Wendland pour une surface *approximante*).
  6. `unsigned int nbIterations` vous permettra de spécifier le nombre d'itérations.
  7. `float radius` vous permettra de spécifier la taille du noyau.

**Exécutez votre code** pour projeter un ensemble de points 3D (par ex, 1000 points) distribués aléatoirement dans le cube  $[-2, 2]^3$ , sur un `pointset`. (*captures d'écran*)

**Créez des résultats pour un noyau Gaussien, avec une taille variable.** Qu'observez-vous? Discutez ce point. (*captures d'écran*)

**Ajoutez du bruit de taille variable (par exemple, d'amplitude  $\in [-\alpha, \alpha]$  dans la direction de la normale, et créez des résultats pour un noyau Gaussien, avec une taille variable.** Qu'observez-vous? Discutez ce point. (*captures d'écran*)