UNIVERSITÉ D'ORLÉANS **M1 Info**
Informatique Année **2016 - 2017**

Série de Travaux Dirigés : Lecteur de trajectoire

Pour ce TD suivre les instructions sous Celene pour récupérer les éléments nécessaires. L'évolution d'une structure moléculaire a été enregistrée au cours du temps. Cette évolution qu'on appelle trajectoire a été stockée dans un fichier et le lecteur de trajectoire consiste donc à afficher successivement toutes les frames enregistrées.

Exercice 1. Lecteur de trajectoire : Version 0

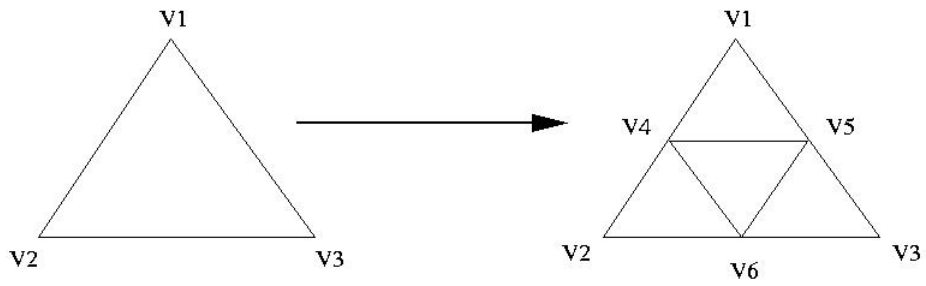
Pour démarrer ce TD vous disposez d'une première version du lecteur de trajectoire. Pour cette version, les atomes sont représentés par des points dont on a augmenté la taille. Sinon les différentes parties de cette version sont

- `xdrfile` dans *Common* qui contient les codes pour lire une trajectoire compressée (les fichiers .xtc). En particulier il contient les fonctions pour lire les positions des atomes d'une frame enregistrée.
- `get_first_frame` dans le fichier `code01_trajectoire.cpp` qui utilise les fonctions de `xdrfile` pour lire la première frame et pour initialiser le tableau des positions des atomes ainsi que le tableau des couleurs.
- `get_frame` qui permet de lire les frames suivantes.
- Les fonctions `init` et `display` qui créent les VBO et VAO nécessaires et réalisent l'affichage sous forme de points.

Exercice 2. Les atomes par des icosaèdres

Désormais on souhaite représenter les atomes par des sphéres. Pour cela on va utiliser des icosaèdres.

L'icosaédre est un polyèdre constitué de 20 triangles définis à partir de 12 sommets. Il est possible de raffiner l'icosaèdre pour se rapprocher de la sphère en subdivisant chaque triangle comme illustrée par la figure ci-dessous.



Dans un premier temps, on va se contenter du premier icosaèdre défini par la variable globale `sommets` et le tableau d'indices correspondant qui sont fournis dans `icosaedre.hpp` dans *Common*.

L'objectif de cet exercice est de modifier le code de la version 0 pour représenter l'atome par un icosaèdre mais en transmettant le minimum de choses à la carte à chaque frame et en faisant le maximum de calculs sur la carte graphique. Pour cela vous allez définir

1. un tableau de sommets qui va être constitué d'autant d'icosaèdres élémentaires que d'atomes. Ce tableau est transmis qu'une seule fois à la carte graphique.
2. un tableau de couleurs qui permet d'associer la bonne couleur à chaque sommet de chaque icosaèdre. Lui aussi n'est transmis qu'une seule fois.
3. un tableau qui permet d'enregistrer la translation à appliquer à chaque icosaèdre pour placer l'atome qu'il représente au bon endroit. Ce tableau contient simplement la position des atomes et est transmis à chaque nouvelle frame à la carte graphique. Pour cela vous devrez utiliser un attribut supplémentaire associé aux sommets.

4. Ecrire le vertex shader qui applique le bon déplacement aux sommets de chaque icosaèdre. Le fragment shader est classique pour attribuer une couleur à partir de l'attribut couleur transmis à la carte.

Remarques supplémentaires

- Les fichiers `petite_trajectoire.xtc` et `trajectoire.xtc` contiennent des trajectoires compressées
- Les fichiers `petite_trajectoire_couleurs.txt` et `trajectoire_couleurs.txt` sont respectivement les fichiers associés à la trajectoire et qui contiennent la liste des atomes qui constituent la structure. Seul le type est indiqué et c'est utile à la fonction `void choix_couleur(const char s, float *rgb)` pour attribuer une couleur (par convention) à chaque atome.
- Ci-dessous un exemple de ce que vous pouvez obtenir pour le moment alors qu'aucun éclairage n'est défini et afin aucun raffinement de l'icosaèdre.