Université d'Orléans — Master Informatique INIS
2017-2018

PROJET N°1
Les Shaders GLSL

---

1 Objectifs

L'objectif de ce projet est de développer un visualisateur de molécules en utilisant un rendu à base de shaders GLSL. Ce projet aura la forme d'un tutorial permettant d'arriver à la solution. Un squelette de programme vous est fourni pour vous permettre de vous concentrer sur les parties importantes de ce projet. Le projet est à faire de manière individuelle.

2 Initialisation : v 0.0

Dans un premier temps, il va falloir modifier le programme fourni afin de pouvoir afficher les molécules en les représentant par des cubes. Le programme qui vous est fourni travaille avec des VBO. Il y a 4 buffers différents :

- `coord_attribute` qui contient l'ensemble des sommets (12 par atomes). Chaque sommet étant représenté par 3 flottants indiquant sa position
- `lesindices` qui contient l'ensemble des triangles à dessiner (20 par atomes). Chaque triangle étant représenté par les 3 indices de ses sommets
- `couleurs_attribute` qui contient la couleur de chacun des sommets (3 flottants pour une couleur)
- `positions_attribute` qui contient la position du centre de l'atome auquel est associé le sommet

Ici, le but est donc de *binder*, dans la fonction `init`, chaque buffer avec sa donnée. Ensuite, il va falloir modifier la fonction `Display` afin de déclencher le rendu de nos cubes/molécules. Les cubes sont tous initialement générés à la même position : ils ont un côté de 1 et sont centrés en `(0,0,0)` (voir fonction `get_first_frame`). Une fois le chargement de **toutes** les molécules, il est donc normal de n'en voir "qu'une seule". Le vertex shader (v1.0) aura pour rôle de positionner les cubes aux bons endroits, en fonction du centre de l'atome.

3 Affichage de sphères

La seconde phase de projet va consister à développer des shaders permettant de rendre une sphère à partir de l'affichage d'un cube.

3.1 Vertex Shader : v 1.0

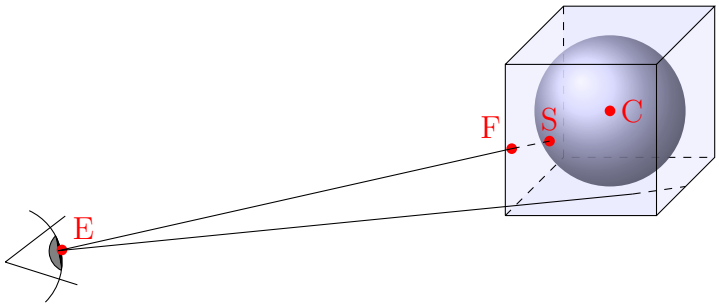
Le vertex shader utilisé par le programme se trouve dans le fichier `VertexShader.vert`.

La première version de ce shader va consister à faire en sorte que les cubes générés s'affichent bien centré sur le point défini par l'attribut de translation `Trans`. Pour cela vous allez rajouter un paramètre d'entrée à votre shader. Ce paramètre devra s'appeler `Trans` et sera un vecteur de 3 flottants. Il est à noter que cette opération est une simple translation du sommet.

3.2 Vertex Shader et Fragment Shader : v 2.0

Le fragment shader utilisé par le programme se trouve dans le fichier `FragmentShader.frag`.

Pour fabriquer la première version de ce shader nous allons devoir faire un peu de géométrie dans l'espace. L'objectif de ce shader est donc d'afficher une sphère en lieu et place du cube normalement rendu par OpenGL. Le schéma ci-dessous illustre ce que nous allons faire.



Le point E représente la position de l'œil (cette position est l'origine du repère de l'œil). Le point F représente la position en 3D du fragment que l'on est en train de traiter et le point S représente l'intersection entre la droite EF et la sphère représentant l'atome associé au fragment.

Les équations en 3D d'une droite de vecteur directeur $v = (a, b, c)$ passant par le point $A = (x_A, y_A, z_A)$ sont :

$$\begin{cases} x = at + x_A \\ y = bt + y_A \\ z = ct + z_A \end{cases} \quad \forall t \in \mathbb{R}$$

ce qui peut s'écrire $X = tv + A$. Par ailleurs l'équation en 3D d'une sphère de centre $C = (x_C, y_C, z_C)$ et de rayon r est $(x - x_C)^2 + (y - y_C)^2 + (z - z_C)^2 - r^2 = 0$. Donc on sait qu'une droite et une sphère s'intersectent si et seulement si le système d'équation à 4 inconnues (x, y, z et t).

$$\begin{cases} x = at + x_A \\ y = bt + y_A \\ z = ct + z_A \\ (x - x_C)^2 + (y - y_C)^2 + (z - z_C)^2 - r^2 = 0 \end{cases}$$

possède une solution. Pour résoudre le système il suffit de commencer par remplacer x, y et z par leur valeur en fonction de t dans la dernière équation. On obtient alors une équation du second degré à une variable (t)

$$(at + x_A - x_C)^2 + (bt + y_A - y_C)^2 + (ct + z_A - z_C)^2 - r^2 = 0$$

en développant cela donne

$$a^2t^2+2a(x_A-x_C)t+(x_A-x_C)^2+b^2t^2+2b(y_A-y_C)t+(y_A-y_C)^2+c^2t^2+2c(z_A-z_C)t+(z_A-z_C)^2-r^2 = 0$$

et refactorisant les termes en t cela donne

$$(a^2+b^2+c^2)t^2+2(a(x_A-x_C)+b(y_A-y_C)+c(z_A-z_C))t+(x_A-x_C)^2+(y_A-y_C)^2+(z_A-z_C)^2-r^2 = 0$$

Si on se rappelle que le produit scalaire de deux vecteurs $(x_1, y_1, z_1).(x_2, y_2, z_2)$ vaut $x_1x_2 + y_1y_2 + z_1z_2$, l'équation peut s'écrire

$$v.v \times t^2 + 2 \times v.(A - C) \times t + (A - C).(A - C) - r^2 = 0$$

Donc c'est seulement dans le cas où cette équation a au moins une solution que la sphère et la droite s'intersectent. Il reste maintenant à trouver le vecteur directeur de la droite EF (c'est simplement $E - F$ si on considère que E et F sont des vecteurs de coordonnées) et un point de la droite. Pour que le fragment connaisse sa position 3D dans les coordonnées de l'œil, il faut que le vertex shader le lui fournisse par l'intermédiare d'un paramètre *varying*. Il en est de même pour le centre de la sphère. Le rayon de la sphère lui est fixé à 1.2.

Modifiez votre vertex shader pour qu'il fournisse au fragment sa position 3D et le centre de la sphère à laquelle il est rattaché. Ecrire le fragment shader qui détruit le fragment quand la droite partant de l'œil et passant par lui n'intersecte pas la sphère.

3.3 Affichage dynamique : v 3.0

L'affichage dynamique permet d'afficher l'évolution (dans le temps) de notre molécule. La fonction `get_frame` permet d'obtenir la position des molécules en $t + 1$. Modifiez la fonction `Idle` afin de charger une nouvelle frame, pour obtenir un affichage dynamique des molécules. Il ne faut pas oublier de *re-binder* les nouvelles positions dans le VBO !

3.4 Fragment Shader et Éclairage : v 4.0

Nous allons maintenant mettre en place un éclairage afin d'obtenir un effet 3D. La fonction *eclairage* fait un éclairage rudimentaire. Elle prend en paramètres une couleur, la position du fragment et la normale du fragment. Pour calculer la normale du fragment, il va falloir résoudre complètement le système d'équations de la section précédente pour trouver le point d'intersection S entre la droite et la sphère. Une fois ce point trouvé, la normale à ce point est $S - C$ où C est le centre de la sphère.

Modifiez le fragment shader pour mettre en place l'éclairage.

4 Rendu

Vous devez rendre, **avant** le **18 mars (23h59m59s)**, une archive contenant :

— Les sources des programmes *obligatoires* (les versions v 1.0, v... dans différents dossiers)

— Une ou plusieurs versions améliorées, le but étant d'explorer les fonctionnalités des shaders. Par exemple :
  — Lumières
  — Déplacement 3D
  — Interaction avec la scène
  — Ombres
  — Textures
  — ...

— Un petit document pdf (ou texte brut) indiquant ce que vous avez implémenté et comment vous l'avez fait.

Quelques petits trucs utiles

— Le produit scalaire de u et v s'écrit `dot(u,v)` en GLSL.

— La fonction `normalize` permet de normaliser un vecteur.

— La fonction `sqrt` calcule la racine carrée.

— La fonction `mult` permet de multiplier une matrice par un vecteur.

— Dans le vertex Shader `gl_Vertex` donne la position 3D du sommet dans le repère de l'œil.

— Dans le fragment Shader `discard;` permet de ne pas afficher un pixel.