1. Interaction avec les shaders

1.1. Contrôle de la dimension de l’objet

On souhaite faire gonfler/dégonfler un objet 3D dynamiquement. Une solution consisterait à utiliser la fonction glm::scale, lui passer un coefficient mis à jour dans la fonction idle mais on peut le faire aussi en passant le coefficient au vertex shader, même si c’est plus lent car plus de travail pour la carte graphique (multiplication par un vecteur pour tous les points). Le but de cet exercice est de montrer comment passer une valeur flottante différente au vertex shader à chaque affichage. On en peut pas directement transmettre une valeur au shader, il faut d’abord interroger le shader pour obtenir un identifiant de la variable du shader, et c’est par cet identifiant que se fait la communication.

Pour obtenir cet effet, on va partir du code utilisant la fonction glm::scale (sur le github) : on déclare une variable globale de type float nommée scale initialisée à 1.0f, on la met à jour dans la fonction idle qui déclenche un nouvel affichage et ainsi de suite.

Vertex shader :
déclarer la variable uniform float scale qui va être mise à jour par le programme principal
modifier le calcul de la position pour que les composantes du point soient multipliées par le même coefficient scale

Code principal :

1. on commente l’appel à glm::scale
2. on ajoute une variable globale unsigned int scaleid qui va contenir l’identifiant de la variable scale du shader
3. à la fin de la fonction initShaders, on récupère l’identifiant de la variable uniform scale du shader à l’aide de la fonction glGetUniformLocation :
   scaleid = glGetUniformLocation(progid, "scale");
4. dans la fonction display, on passe la valeur de scale au shader avant l’appel à glDrawElements :
   glUniform1f(scaleid, scale) ;

Et ça devrait fonctionner !

Vous pouvez modifier le code pour :

- que l’effet soit moins brutal en utilisant une fonction trigonométrique,
- que la mise à l’échelle soit contrôlée au clavier,
- …

1.2. Inversion des couleurs

On veut activer l’inversion des couleurs lors de l’appui sur une touche du clavier. Pour cela on passe une valeur booléenne au vertex shader : si invert alors on inverse sinon on laisse la couleur par défaut.
Code principal :

1. définir une variable globale bool invert = false;

2. définir une variable globale unsigned int invertid;
3. récupérer l'identifiant dans initShaders comme dans l'exercice précédent
4. définir une fonction keyboard :
   void keyboard(unsigned char key, int x, int y)
5. dans cette fonction ajouter une conditionnelle : si key == 'i' alors invert = !invert
6. dans le main, passer la fonction de gestion du clavier à GLUT :
   glutKeyboardFunc(keyboard);
7. dans la fonction display ajouter la ligne suivante avant l'appel à glDrawElements :
   glUniform1i(invertid, invert);
   Note : en C++ les booléens sont des entiers on utilise donc le suffixe 1i pour la fonction glUniform.

2. Éclairage

Jusqu'ici, les objets possèdent leurs couleurs propres, et il est possible d'associer une couleur à chaque point d'un objet. Cela peut être utile pour des environnements non réalistes mais dans un univers 3D on souhaite généralement avoir un éclairage réaliste dépendant de la position du soleil virtuel, de lumières positionnées dans la scène, etc. L'éclairage effectué par les shaders est entièrement paramétrable. On s'intéresse ici à l'éclairage dit de Phong qui est constitué de 3 composantes :

1. ambiante : la lumière qui enveloppe tout le monde 3D,
2. diffuse : la lumière provient d'une source à une certaine position du monde 3D et seulement les faces de objets orientées face à cette source sont éclairées,
3. spéculaire : suivant la position de la source de lumière et de l'observateur certaines faces seront plus éclairées que les autres.

L'éclairage dépend donc des caractéristiques de la lumière : position et couleur, et également du matériau des objets éclairés. Un objet vert éclairé en rouge va donc apparaître jaune. Dans la suite on n'utilise donc plus la couleur des points mais on attribue une couleur globale à l'objet. Pour plus de détails on utilisera ensuite des textures. Par contre pour déterminer si une face d'un objet est plus ou moins éclairée, il faut envoyer les normales à chaque point.

Pour comprendre comment cela fonctionne on part du code disponible sur le dépôt github qui contient les bases pour l'éclairage de Phong mais avec des informations (couleurs, position de la lumière) en dur que l'on va sortir pour pouvoir les contrôler du code principal.