Objet: Réaliser un shader de Phong en plusieurs étapes:

Créer un répertoire TPIllumination. Dans ce répertoire et pour chaque étape, créez un répertoire etapex.

Étape 1 : Premiers pas avec les shaders

Télécharger le code source de départ sur plubel "code TP Illumination". Décompressez le dans "TPIllumination".

Le code contient la préparation de l'affichage d'un tore texturé avec : VAO, VBO et shaders. Pour finaliser l'affichage il suffit de compléter les shaders. L'envoi des variables aux shaders est fait pour cette étape. Prenez connaissance du code (aucun commentaire n'est permis sur la qualité de la structuration du code, i.e. variables globales,... ;).

Dans vertshader:

- Récupérer la position du sommet
- lui appliquer la transformation Model/vue/Projection (MVP)
- transmettre (out FragPosition) au fragment shader
- Dans le fragment shader : afficher la forme avec une couleur uniforme (définir une variable out vec4 finalColor) lui affecter une constante (par exemple vec4(.5,.5,0.,1.))
- testez l'affichage
- Modifier le fragment Shader en affectant à la couleur finale le vecteur position du fragment (Attention ajoutez une composante pour Alpha).
- testez l'affichage

Étape 2 : Lumière ambiante

Dans le code opengl:

- Passer aux shaders la valeur de la variable LightAmbientCoefficient : définissez une variable uniform (comme champ d'une structure "Light", utilisez la fonction locLightAmbientCoefficient = glGetUniformLocation(programID, "light.ambientCoefficient");
- locLightAmbientCoefficient récupere la "location" de la variable uniforme qui sera utilisée pour l'envoi de la valeur au GPU (point suivant)
- envoyer au GPU la valeur de la variable fonction :

```
glUniform1f(locLightAmbientCoefficient,LightAmbientCoefficient);
```

- idem avec la variable LightIntensities qui est un vecteur à 3 composantes qui représente la couleur de la source lumineuse.

Dans fragment shader :

- Définissez la structure Light et la variable light :

```
uniform struct Light {
   vec3 intensities;
   float ambientCoefficient;
} light;
```

- Calculez une variable "ambient" = coeffambiant × intensité de la source lumineuse × couleur de la surface .
  (pour l'instant vous affecterez une couleur constante , mais bientôt on mettra la couleur de la texture à la place).
- l'affecter à la couleur du fragment.
- Testez l'affichage (touche 'A' pour augmenter le coeffAmbient , 'a' pour le diminuer.

Étape 3 : Lumière diffuse, c'est déjà trop beau!

Nous avons besoin de calculer de produit scalaire entre la normale à la surface et le vecteur directeur de la surface à la lumière.

Remarque : Dans le contexte des shaders il n'est pas nécessaire d'activer les lumières d'opengl, car tout se passe dans notre shader,i.e. on peut gérer notre structure de données comme on veut et en faire ce qu'on veut (youpi!), et c'est ce qu'on fait ici.

Dans le code opengl, il faut passer les informations pour faire le calcul :

- la position de sommet (déjà fait via un VBO)
- la normal (idem)
- la position de la lumière (à mettre dans la structure Light).
- il faut passer la matrice de transformation du modèle, car la normal est définie dans le repère de l'objet, il faudra la recalculer après transformation du modèle.

Dans le fragment shader:

- il faut récupérer la normale par la "location" correspondante (voir code opengl) et la faire suivre aux fragment shader : définir une variable out et lui affecter la valeur de la normale.

Dans fragment shader :

- déclarer la variable uniform pour la matrice de transformation du model
- Calculer la normale transformée :

```
vec3 normal = normalize(transpose(inverse(mat3(MODEL))) * fragNormal);
```

  – "fragNormal" est la la normale interpolée envoyé du vertex shader pour la fragment. À déclarer en in vec3 fragNormal ;

  - et MODEL est la variable uniforme passée depuis le code opengl.

- Calculez la position du point de la surface, correspondant au fragment, dans la scène, i.e. après application de la transformation du modèle. Je vous laisse trouver comment faire.
- Calculez le vecteur (que l'on peut normer) de la position du fragment (dans la scène) à la lumière (surfaceToLight).
- Calculez le coefficient diffuseCoefficient = max (0 , normal • surfaceToLight));
- et enfin la composante de la lumière diffuse :
  lumiere diffuse = diffuseCoefficient × courleur de la surface × light.intensities;
- N'oubliez pas d'ajouter la composante de Lumière diffuse à la couleur de fragment
- Testez en déplaçant la lumière touche 'x','X','y', 'Y','z' et'Z' + autres paramètres (voir le code).

Étape 4 : Lumière Spéculaire, là on s'y croirait!

Maintenant que vous êtes rodés, je simplifie les explications. Pour la lumière spéculaire nous devons calculer le coefficient de lumière spéculaire :

$$specularCoefficient = max(0, < surfaceToCamera, ReflectLight >)^{materialShininess}$$

où ReflectLight est la direction de la lumière réfléchie par la surface , i.e. le vecteur direction de la lumière symétrique par rapport à la normale. On obtient par :

```
reflect(-surfaceToLight, normal)
```

Attention si le coefficient de lumière diffuse est négatif ou nul le coefficient de lumière spéculaire doit être nul.

Ensuite la contribution spéculaire est le produit du coefficient précédent par la couleur spéculaire du matériau par l'intensité de la lumière (couleur de la lumière).

Il faut modifier le fragment shader, mais avant il faut envoyer au shader toutes les données nécessaires (dans le code opengl).

Étape 5 : Avec coefficient d'atténuation, là c'est top !

Pour faire encore plus vrai on ajoute une atténuation

On calcule a distance de la source lumineuse à la surface et le coefficient d'atténuation est donnée par :

```
attenuation = 1.0 / (1.0 + light.attenuation * pow(distanceToLight, 2));
```

où light.attenuation est le coefficient d'atténuation de la source lumineuse. Le coefficient d'atténuation ainsi calculé s'applique à la somme de la composante diffuse et de la composante spéculaire.

Étape 6 : correction gamma, là c'est la grande classe

Permet de compenser le fait que l'écran ne restitue pas les couleurs de façon linéaire (il faut ajuster la valeur de gamma à son écran) la composante linéaire est ce qu'on calcule : linearColor = ambient + attenuation*(diffuse + specular);

La correction se fait par la formule suivante sur les 3 composantes de la couleur (sauf le canal alpha).

$$finalColor = linearColor^{gamma}$$

Étape 7 : la color map, là on ne sait plus quoi dire

Appliquez la texture à la place de la couleur uniforme. Pour cela on utilise la fonction texture(nom_de_la_variable_uniform_de_la texture, coordUV)

Étape 8 : Enfin la normal map, là plus rien à ajouter

On récupère un vecteur dont les coordonnées sont entre 0 et 1 (car couleur). On les recale entre -1 et 1 (dur dur) et on l'ajoute à la normale à la surface avant transformation.

Étape 9 : ben si, la glossy Map !

Dans la vraie vie, les matériaux non pas des caractéristiques de "brillance" uniformes. La glossy Map permet de définir là où le matériau est grillant là ou il l'est moins. Il suffit d'utiliser ma texture comme coefficient spéculaire local (K_s), effet garanti.

Étape 10 : mais quand on s'arrête?

Ici pour ce tuto, mais en fait jamais, car on peut encore faire

- du Displacement map mais il faut attaquer le tesselation shader, et oui il y à d'autres.
- les ombres portées
- les reflets des miroirs
- utiliser des BRDF
- ...