Programmation Graphique Haute Performance
Éclairage local

March 21, 2016

Objectifs pédagogiques : calcul des normales aux sommets d'un maillage, calcul de l'éclairage local à l'aide de différentes BRDF et méthodes d'interpolation.

Objectifs applicatifs : rendu d'une scène 3D à l'aide d'un modèle d'éclairage local, et de sources lumineuses directionnelles.

1 Prise en main

Téléchargez et compilez le nouveau projet. Procédez de la même manière que dans le TP précédent.

2 Gestion des normales

Pour pouvoir calculer l'éclairage, nous avons besoin de connaître l'orientation de la surface. Cette information est représentée par les normales (vecteurs 3D) que nous noterons **n**. Les normales sont transmises par la classe `Mesh` aux shaders en tant qu'attributs `vec3 vtx_normal`.

1. Modifiez le vertex shader `simple.vert` pour qu'il récupère l'attribut `vtx_normal` en entrée et qu'il le transmette au fragment shader. Modifiez ensuite le fragment shader de façon à afficher les normales sous forme de couleurs. Attention, les coordonnées des normales sont comprises dans l'intervalle [-1;1], il faut les transformer dans l'intervalle [0;1] pour les afficher comme des couleurs. Testez avec le maillage fourni, dont les normales sont pré-calculées et stockées dans le fichier.

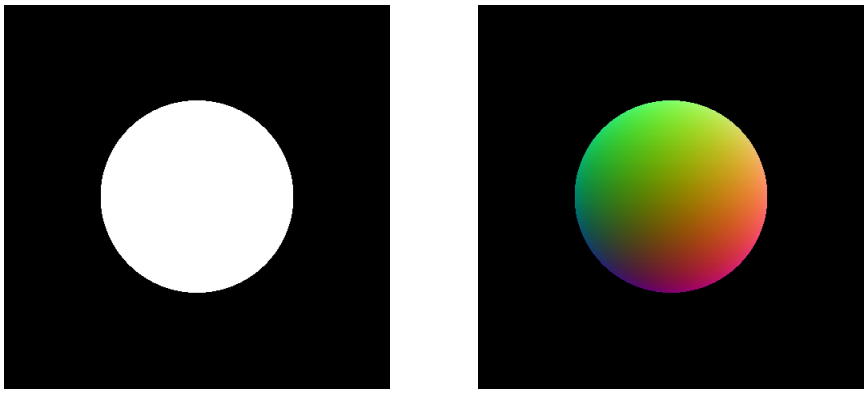

Figure 1: Sphere avec le rendu de base (gauche), et avec le rendu par normal (droite).

2. Les modéles 3D ne sont pas toujours fournis avec leurs normales.
Complétez la méthode `void Mesh::computeNormals(...)` pour qu'elle calcule les normales en fonction de la position des sommets. Testez avec le maillage `tw.obj` (dans la méthode `Viewer::init(...)`).

Rappels : La normale d'un sommet est obtenue en faisant la moyenne pondérée des normales des faces adjacentes à ce sommet, sachant que la normale $\boldsymbol{n}_f$ d'une face de sommets $(\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3})$ est :

$$\boldsymbol{n}_f = (\boldsymbol{v_2} - \boldsymbol{v_1}) \times (\boldsymbol{v_3} - \boldsymbol{v_1})$$

où $\times$ représente le produit vectoriel. Il s'avère que $\|\boldsymbol{n}_f\| = 2\Delta$, où Δ est l'aire du triangle $(\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3})$. Par conséquent, en calculant directement la moyenne des normales $\boldsymbol{n}_f$ des faces adjacentes à un sommet, nous obtenons une moyenne *pondérée par les aires* des triangles adjacents. Cette pondération est fréquemment utilisée car elle produit de bons résultats tout en étant rapide à calculer.

Indices : Si votre implémentation utilise des boucles imbriquées ou une structure de donnée temporaire, vous faites fausse route. L'algorithme consiste à calculer les normales en trois passes : (1) initialiser toutes les normales au vecteur nul, (2) calculer les normales des faces et les ajouter à leurs trois sommets incidents au fur à mesure, et (3) normaliser la normale de chaque sommet.

3. Les normales transmises aux shaders sont définies en espace objet. Or, pour le calcul de l'éclairage, nous avons besoin de connaître l'orientation de la normale par rapport à la direction de la lumière incidente et la direction de vue. Pour cela, il faut exprimer tous ces vecteurs dans le même repère. Ajoutez une variable `mat4 normalMatrix` au fragment shader, et initialisez là correctement. Lorsque vous bougez la caméra, la couleur des normal doit maintenant varier en fonction du point de vue.

Rappels : La matrice de transformation des normales peut se déduire de la matrice de vue grâce à la formule suivante :

$$\boldsymbol{N} = (\boldsymbol{L}^{-1})^\top$$

où $\boldsymbol{L}$ est la partie linéaire de la matrice de vue (sous-matrice 3×3 supérieure gauche).

3 Shading

3.1 Modèle d'illumination de Blinn-Phong

Vous allez utiliser la BRDF *Blinn-Phong* pour calculer l'éclairage local. C'est un modèle *empirique* qui est efficace à calculer et produit un résultat plausible, mais qui n'est pas physiquement réalistes.

1. Modifiez les shaders `simple.vert/.frag` pour pouvoir calculer l'éclairage avec la BRDF de *Blinn-Phong* pour une source lumineuse directionnelle. Voici le prototype de la fonction principale :

```
/**
 *  n : la normale de la surface
 *  l : un vecteur pointant vers la source de lumière
 *  v : un vecteur pointant vers la caméra
 *  diffuse_color : la couleur diffusée par l'objet
 *  specular_color : la couleur du reflet spéculaire
 *  exponent : le coefficient speculaire de la BRDF de Phong
 *  light_color : la couleur de la lumière
 */
vec3 blinnPhong(vec3 n, vec3 l, vec3 v, vec3 diffuse_color, vec3 specular_color,
           float exponent, vec3 light_color) {
  return vec3(.5);
}
```

Procédez par étape :

a. Commencez l'implémentation de la fonction `blinnPhong` de façon à calculer uniquement la **partie diffuse** de la BRDF. Rappelons qu'il s'agit d'une constante ρ_d, qui correspond ici à la variable `diffuse_color`. N'oubliez pas de prendre en compte le terme en cosinus (produit scalaire entre la normale et la direction vers la lumière) et la couleur de la lumière.

   Pour tester, fixez dans votre shader la direction de la lumière `l` à une valeur arbitraire, par exemple `normalize(vec3(1.))`. Dans un premier temps, utilisez une couleur uniforme pour l'objet, par

exemple `vec3(.5)`. De même pour la couleur de la lumière, fixez d'abord sa valeur à `vec3(1.)`. Appelez la fonction `phong` avec ces paramètres pour obtenir la couleur de sortie, et transmettez-là au fragment shader.

b. Rajoutez la **partie spéculaire** de l'éclairage, qui simule un matériau brillant. Celle-ci s'obtient par la formule $\rho_s(\boldsymbol{h} \cdot \boldsymbol{n})^\eta$, où $\boldsymbol{h}$ est le demi-vecteur entre le vecteur de vue `v` et celui de la source lumineuse `l`, ρ_s correspond au paramètre `specular_color` et η correspond au paramètre `exponent`. Testez avec `specular_color = vec(1.)` et avec un `exponent` assez élevé, par exemple 128.

4 Aller plus loin

- Pour le moment, la lumière de la scène est une lumière directionnelle, représentée par un simple vecteur de direction. Modifiez le code pour simuler une lumière ponctuelle émettant dans toutes les directions. L'intensité lumineuse diminue en fonction du carré de la distance.

- Modifiez le code pour simuler un spot lumineux. Le spot est caractérisé par une direction, et un angle d'ouverture.