INFSI 350 – Réalité Virtuelle
Travaux Pratique

Rendu

L'archive du TP sur trouve sur :
http://www.telecom-paristech.fr/~boubek/ens/rv/tp/infsi350_tp_rendu.zip

I. Lumière, matériaux et textures en OpenGL

Pour cette partie, on partira du code du progamme gmini, disponible dans l'archive du tp. On s'appuiera sur la documentation disponible sur le site www.opengl.org.

I.a Activer plusieurs lumières, de couleurs différentes. Lier ces options à des touches clavier.

I.b Dessiner en rouge les triangles éclairés par la première lumière, en bleu les autres. Attention, il faudra changer de couleur à chaque face et non à chaque sommet.

I.c Comment définir les matériaux en OpenGL ? Définir plusieurs matériaux et rajouter une clé clavier permettant de passer de l'un à l'autre sur le modèle géométrique courant.

I.d Désactivez les lumières OpenGL (glDisable (GL_LIGHTING). La couleur des sommets est désormais uniquement dépendante des couleurs spécifiées aux sommets (glColor3f). Implémenter la BRDF de Phong et utilisez-la pour calculer une couleur à chaque sommet. Attention, il faudra définir une ou plusieurs sources de lumières.

I.e L'archive du TP contient un ensemble de textures (carte couleurs stockée sous forme d'images dans notre contexte) dans le répertore « Textures ». Ces images sont au format PPM (http://netpbm.sourceforge.net/doc/ppm.html) et peuvent visualisée avec gimp (http://www.gimp.org).

Rajouter une fonction de chargement de ces textures dans votre programme OpenGL. La carte de couleur devra être lu depuis l'une des textures, stockée dans un tableau et passée à OpenGL. Utilisez cette texture sur un objet simple. Attention, il faudra activer les textures OpenGL et spécifier, pour les chaque sommet, avant la position (glVertex3f) les coordonnées de texture (paramétrisation UV) à l'aide de l'opérateur glTextCorrd2f. Ci-dessous un morceau de code pour cette mise en place (int peut être remplacé par Glint, et unsigned int par Gluint):

```
void init () {
  ...
   unsigned int texMap;
   unsigned int texMapLevel;
   unsigned int texMapWidth;
   unsigned int texMapHeight;
   unsigned int texMapComponents;
   unsigned int texMapFormat;
   bool texMapGenerated;
   glGenTextures (1, &texMap);
   glBindTexture (GL_TEXTURE_2D, texMap);
   glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
   glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
```

```
   glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   gluBuild2DMipmaps (GL_TEXTURE_2D, texMapComponents, texMapWidth, texMapHeight,
texMapFormat,
                GL_UNSIGNED_BYTE, texMapData);
   ...
}

void display () {
    glEnagle (GL_TEXTURE_2D);
    glBindTexture (GL_TEXTURE_2D, texMap);
    ...
    for (...) {
        ...
        glTexCoord2f (u, v);
        glVertex3f (x, y, z));
        ...
    }
}
```

II.Ambient Occlusion

Dans cette partie, on implémente l'algorithme d'*Ambient Occlusion* statique. L'idée de précalculer une couleur par sommet indiquant la proportion d'occlusion présent dans le voisinnage. L'algorithme sera lancé lorsque la touche 'o' sera pressée

II.a Désactiver l'éclairage OpenGL et tester la fonctionnalité de couleur par sommet (glColor3f) en remplissant du noir vers le blanc les sommet en fonction de leur distance à la caméra. Il faudra définir un attribut couleur par sommet.

II.b Définir une class « Ray » et implémenter un méthode testant l'intersection d'un rayon et un triangle.

II.c Pour chaque sommet, distribuer aléatoirement N rayons autour de son vecteur normal (hémisphère visible) et évaluer la valeur d'occultation (cf cours Ambient Occlusion) en comptant le nombre d'intersection avec les triangles situés dans une sphère de rayon R. Utiliser cette valeur pour définir une couleur (e.g. Niveau de gris) au sommet. Vous pouvez définir différent paramétres (liés à des touches clavier) pour contrôler ce processus: nombre de rayons N, angle du cone de distribution des rayons, distance maximale R du test d'intersection, divers fonctions « AO vers couleur ». Attention à la robustesse de vos tests. Exemple: s'assurer que les rayon ne compte pas pour intersection valide le point de départ. Quelle limitation général y a-t-il avec l'algorithme proposé ? Quelle est la complexité de votre algorithme ? Comment améliorer cette complexité ?

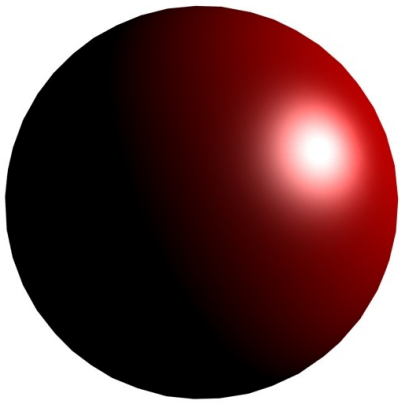
II.d Combiner éclairage OpenGL et Ambient Occlusion.

Bonus

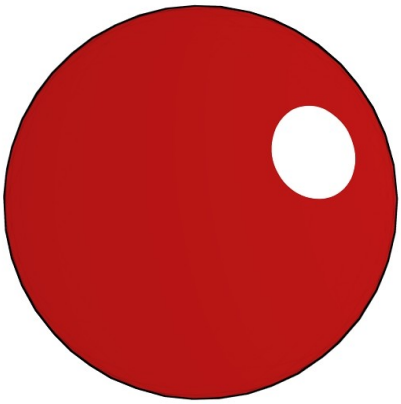
Comment calculer ou approximer l'Ambient Occlusion plus rapidement ? Implémenter votre solution.

III. Programmation GPU pour le rendu.

Dans cette partie, il faudra utiliser le programme gmini-sl, disponible dans le répertoire de l'archive du TP. Attention, la bibliothèque GLEW et un GPU supportant OpenGL 2.0 ou plus sont nécessaire.



Ombrage de Phong



Ombre NPR type *cartoon*

III.a Compiler le programme et utilisez-le pour charger un objet. L'objet est tout blanc car la fonction GLSL régissant la couleur des pixels (dans le code du fragment shader « shader.frag ») place le blanc comme valeur par défaut de tous els pixels. Observez le code de chargement des shader dans le pogramme C++ ainsi que le code GPU pour le vertex (shader.vert) et le fragment shader (shader.frag).

III.b Implémentez la BRDF de Phong dans le fragment shader. Notez que vous disposez de la position (P) et de la normal à la surface (N) pour chaque fragment.

III.c On souhaite maintenant produire un rendu non photo-réaliste (NPR) de type « cartoon ». Pour se faire, le fragment shader doit générer de grands applats de couleur unis. L'idée est de choisir trois couleurs (blanc pour le spot spéculaire, arbitraire pour la couleur des objets et noir pour les contours). Implémentez ce rendu au niveau du fragment shader. Votre fonction doit pouvoir discriminer les fragments afin de leur affecter l'une des trois couleurs.