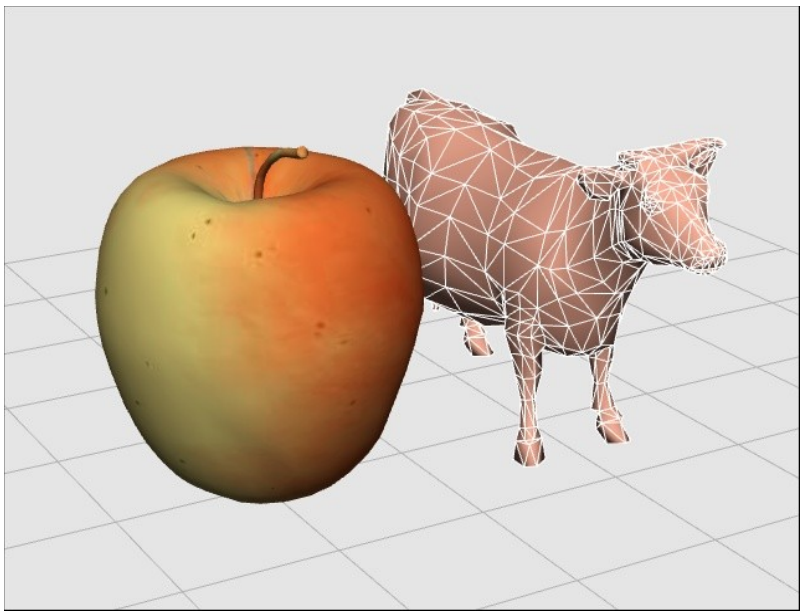
TP n°4 : Matériaux et Lampes

Le but du TP est de comprendre comment fonctionnent les matériaux et l'éclairement par des sources de lumière. Il montre aussi comment construire une classe pour représenter un matériau en relation avec les maillages vus la semaine dernière.

Téléchargez l'archive du TP4 qui est sur Moodle et décompressez-la. Comme chaque fois, utilisez le script `server.py` pour afficher les pages, à cause des dossiers `libs` et `data`.

1. Étude de la classe Material : 01 – MeshMat

Ce projet est un bac à sable tout prêt, vous n'avez qu'à comprendre comment il fonctionne et vous pourrez tester des modifications.



Le cours explique différents aspects de la classe `Material`. Vous allez devoir les étudier. Son source est dans le dossier `libs` car elle est utilisée dans tous les projets du TP.

Cette classe est la super-classe de tous les matériaux. Quand on définit une sous-classe, on doit au moins surcharger le constructeur parce que c'est lui qui définit les deux sources des shaders, vertex et fragment. Allez voir la classe `MaterialEdge`, c'est la plus simple. Vous remarquerez que le constructeur de sa super-classe est appelé à la fin. C'est possible en JavaScript ; par contre tant que `super()` n'a pas été appelée `this` n'existe pas.

Le constructeur de `Material` compile le shader puis va chercher certaines variables `attribute` associées aux VBO et `uniform` associées aux matrices. On emploie toujours les mêmes. Elles ne sont évidemment pas toutes utilisées dans chaque shader ; c'est prévu, leurs identifiants restent à `null`.

La classe `Material` définit deux méthodes pour activer le matériau et le désactiver en fin de dessin. Ce sont les instructions qui étaient avant dans la classe `Mesh`. Du coup, dans sa méthode `onDraw` il ne reste plus grand-chose que l'activation du VBO des indices et le dessin des triangles ou des lignes. J'ai compliqué pour permettre de dessiner les arêtes

d'un maillage mais vous pourriez enlever tout ça. Pour empêcher de dessiner les arêtes, il suffit, dans la classe `Cow`, de ne pas créer de matériau pour elles.

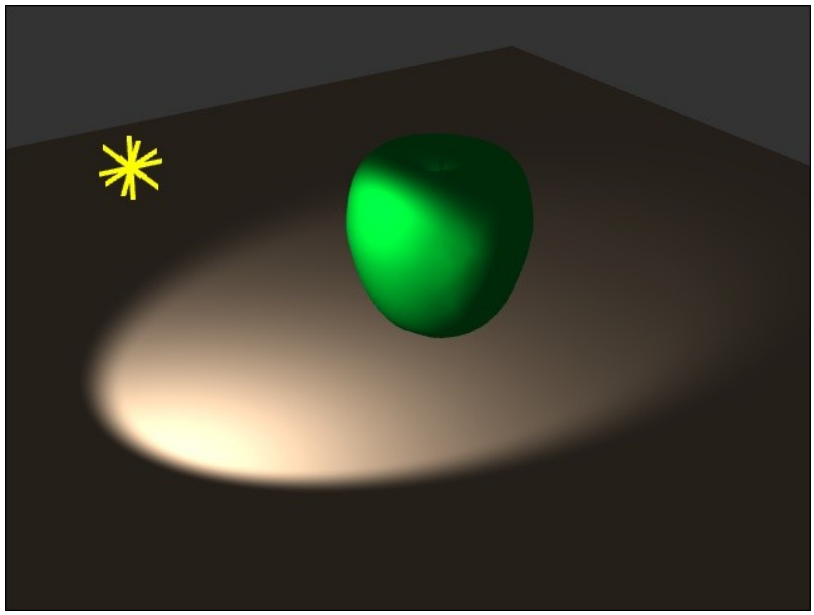
Les méthodes `select` et `deselect` de `Material` doivent être surchargées lorsque le matériau fait quelque chose de spécifique, par exemple changer la largeur des lignes, ou activer une texture. Dans ce cas, il faut impérativement appeler la méthode de la superclasse. Même chose pour la méthode `destroy`.

Les shaders de `MaterialCow` et `MaterialApple` calculent un éclairement diffus type Lambert. C'est l'objet de toute cette semaine. Dans le cas de la pomme, il fait appel à une texture. Ne creusez pas pour l'instant, il y a un autre exercice pour ça.

Maintenant, vous pouvez regarder les sources de `Apple` et `Cow`. Ca devient assez simple à comprendre car tout est mis dans les deux classes `Mesh` et `Material`. Vous remarquerez que le matériau doit être créé avant d'appeler le constructeur de `Mesh`, du coup, on le met dans une variable locale car `this` n'existe pas encore ; par contre, on le reprend dans une variable d'instance dès que c'est fait.

2. Réalisation de lampes : 02 – Lampes

Ce projet permet de mettre en œuvre différents types de lampes. Tout se passe dans la classe `MaterialColor`. Les lampes n'existent pas en réalité en tant qu'objets. C'est seulement les calculs de couleur qui donnent l'impression que la surface est éclairée.



Actuellement, les calculs simulent une lampe située à l'infini, ayant une direction constante comme le soleil. Le matériau est seulement diffus, « lambertien ». Le résultat est moyen sur le plan parce que l'angle ne changeant pas, ça donne la même couleur.

La classe `Light` permet de stocker les informations de la lampe. La classe `Scene` initialise une instance. La position de la lampe est donnée en coordonnées homogènes. Comme w=0, c'est une lampe directionnelle, donc en fait ce n'est pas une position mais un vecteur donnant une direction. La méthode `transform` permet de calculer les coordonnées dans le repère caméra.

Votre travail consiste à remplacer cette lampe par une positionnelle en (-3.0, 3.0, 1.0, 1.0), la ligne est commentée dans `Scene`. Cela consiste à modifier l'initialisation dans la scène et le calcul du vecteur L dans le shader.

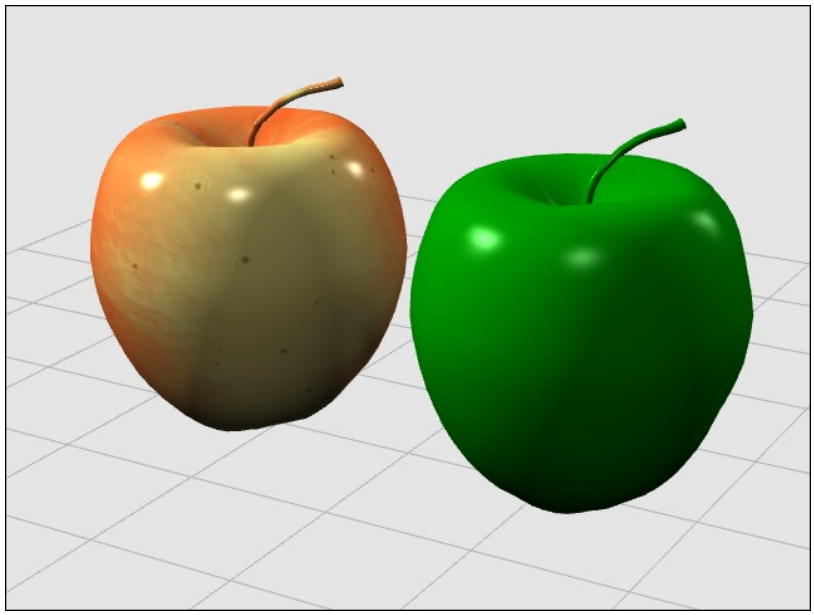
Ensuite, faites en sorte que l'intensité de l'éclairement décroisse en fonction du carré de la distance entre la lampe et le fragment. Vous aurez alors à modifier l'initialisation de la couleur de la lampe, car elle sera trop sombre : mettez 15 pour chaque composante.

Le travail suivant consiste à définir une lampe spot pour obtenir l'effet de la copie-écran ci-dessus. Pour cela, il faut rajouter des informations à la lampe, donc à la fois dans les classes `Scene`, `Light` et `MaterialColor`. Une lampe spot est à la fois positionnelle et directionnelle, donc la variable qui donne la position ne suffit pas. En plus, elle gère deux angles : celui du plein éclairement appelé `MinAngle` et celui de l'extinction hors faisceau appelé `MaxAngle`.

- Classe `Scene` :
  - `this.m_Light.setDirection(1.5, -2.0, -0.5, 0.0);`
  - `this.m_Light.setAngles(30.0, 38.0);`
- Classe `Light` :
  - il faut rajouter une variable d'instance `m_LightDirectionScene` et `m_LightDirectionCamera`, comme les `m_LightPosition*` existantes. Dupliquez et éditez les setter et getter. La méthode `transform` doit aussi traiter cette variable et en plus, elle doit normaliser la direction en coordonnées caméra, pour éviter au shader de le faire.
- Classe `MaterialColor` :
  - Rajoutez les variables *uniform* qui reçoivent les nouvelles informations : direction et les deux angles. Alors en fait, il est préférable de fournir directement les cosinus de ces angles au shader plutôt que le laisser les calculer pour chaque fragment.
  - Effectuez le calcul indiqué dans le cours pour déterminer si le fragment est dans le cône d'éclairement ou pas.

Comment pourrait-on faire pour que la lampe soit attachée à la caméra au lieu de la scène ? C’est à dire qu’en tournant autour de la scène, l’éclairage ne changerait pas de direction, comme si on tenait la lampe à la main.

3. Modèles d'éclairement : 03 – Phong



Ce projet va vous permettre de coder différents modèles d'éclairement, diffus et spéculaires.

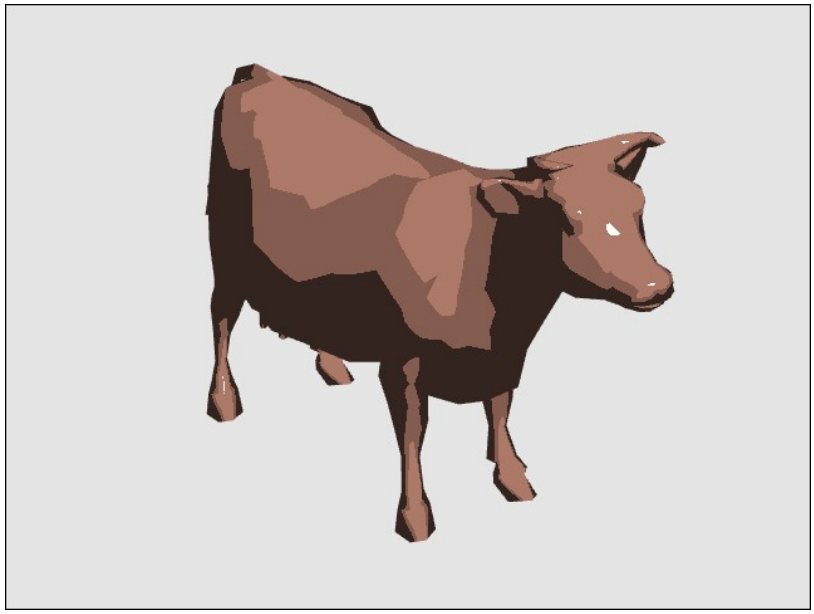
La classe `Scene` crée trois lampes. Elles sont transmises aux deux matériaux. Leurs shaders déclarent des tableaux de trois vecteurs pour contenir les couleurs et les positions homogènes des lampes. Dans la méthode `setLights`, c'est bas niveau, il faut recopier les coordonnées et couleurs des lampes dans des tableaux « tassés » et les envoyer au shader.

Les caractéristiques des matériaux sont fixes. Pour la pomme de gauche, la couleur diffuse Kd est définie par une texture.

Votre travail consiste à compléter le code pour calculer l'éclairement diffus et l'éclairement spéculaire. Vous pouvez commencer par les grands classiques : Lambert avec Phong pour la pomme verte, et faire une autre combinaison : Minnaert ou Oren-Nayar avec Blinn pour l'autre pomme. Faites une copie du dossier pour chaque combinaison ou alors dupliquez les matériaux et rajoutez des pommes pour chacun.

4. Éclairement non réaliste : 04 – Toon

Le « Toon shading » consiste à utiliser une fonction constante par morceau au lieu d'une fonction linéaire, pour calculer l'éclairement final. Cela revient à limiter le nombre de couleurs employées afin de faire des plages constantes. Définissez un matériau « MaterialToon » et appliquez-le à l'une des pommes ou sur la vache.



Déposer un fichier zip contenant votre dossier du TP4, avec chacun des projets clairement nommé.

Important : faites extrêmement attention aux chemins des dossiers libs et data présents dans l'archive. Vous devez faire en sorte que votre projet fonctionne tel quel, sorti de l'archive, sur une autre machine et système que le vôtre. Le correcteur ne peut pas consacrer 10 minutes à éditer chacun de vos sources pour qu'il fonctionne.