Modélisation et Synthèse d'Image
TP4 : Shading (2 scéances)

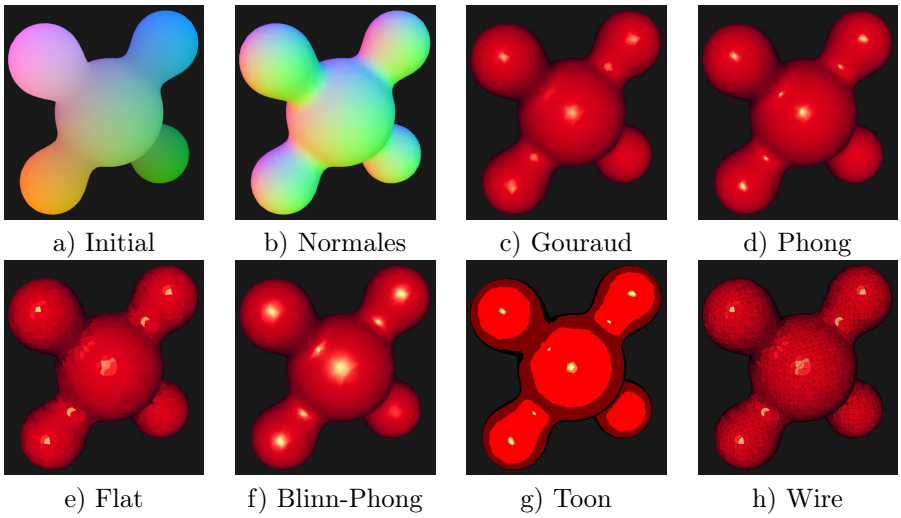


Figure 1: Les différents shadings proposés dans ce TP

Introduction

L'objectif du TP est d'explorer différentes techniques d'ombrage (shading en anglais) un objet 3D.

La base de code qui vous est fournie reprend les éléments de la dernière séance. Pour la récupérer, allez à l'adresse suivante : http://romain.vergne.free.fr/blog/?page_id=228.

Pour l'utiliser, suivez la démarche habituelle :

- extrayez l'archive du TP dans le dossier de votre répertoire personnel dédié aux TP de 3D (`unzip TP4.zip -d ~/TP3D/`)
- accédez au dossier du TP (`cd ~/TP3D/TP4/`)
- créez un lien symbolique vers le dossier `external` (`ln -s ../external/`)
- créez un lien symbolique vers le dossier `models` (`ln -s ../models/`)
- créez un dossier pour la compilation (`mkdir build`)
- accédez à ce dossier (`cd build`)
- lancez `cmake` (`cmake ..`)
- lancez la compilation (`make`)
- exécutez (`./polytech_ricm4_tp4`)

Vous devriez avoir la hiérarchie de fichiers ci-contre.

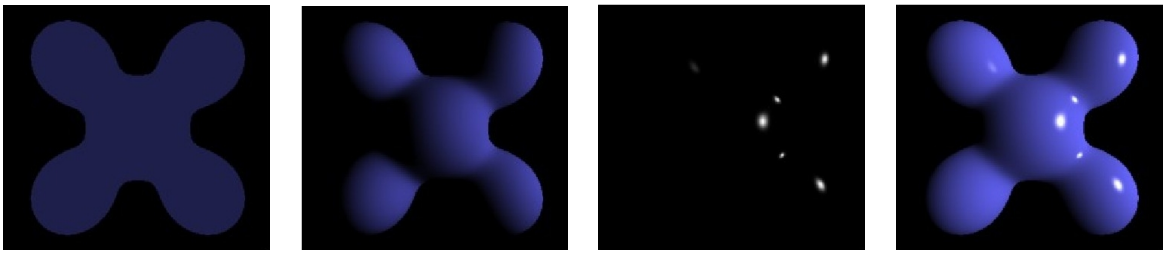
```
TP3D
├── external
│   └── ...
├── models
│   └── ...
├── TP1
│   └── ...
├── TP2
│   └── ...
├── TP3
│   └── ...
├── TP3 bis
│   └── ...
└── TP4
    ├── ...
    ├── external [-> ../external/]
    ├── models [-> ../models/]
    ├── build
    │   └── ...
    ├── shader
    │   └── ...
    ├── src
    │   └── ...
    └── CMakeLists.txt
```

Modèle de Phong

Un modèle d'illumination vise à calculer une couleur à partir d'un ensemble de paramètres d'entrée :

- un ensemble de paramètres relatifs à la surface de l'objet :
  - une position dans l'espace
  - une normale à la surface en ce point
  - une couleur de la surface en ce point
  - une BRDF (Bidirectional Reflectance Distribution Function)
- un ensemble de paramètres relatifs à la lumière :
  - une position dans l'espace ou une direction d'illumination
  - une couleur
  - une intensité

La modèle le plus simple et le plus répandu est le modèle de Phong. Il s'exprime comme la somme de diférentes composantes, comme nous pouvons le voir sur la figure suivante :



Ambiant Diffus Spéculaire Total

Figure 2: Les différentes composantes du modèle d'illumination de Phong

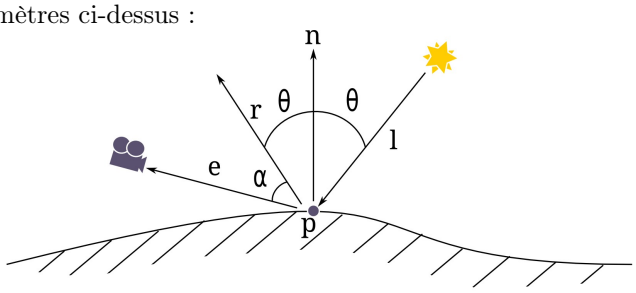
Plus précisément, on note :

$$\underbrace{\rho_a.L_a}_{\text{Ambiant}} + \underbrace{\rho_d.L_d.max(-\mathbf{n}.\mathbf{l},0)}_{\text{Diffus}} + \underbrace{\rho_s.L_s.max(\mathbf{r}.\mathbf{e},0)^s}_{\text{Spéculaire}} = \underbrace{L_f}_{\text{Total}}$$

où :

- ρ_a, ρ_d, ρ_s, sont les coefficients associés à chaque composante
- L_a, L_d, L_s, sont les couleurs de chaque composante
- L_f est la couleur résultante
- n est la normale de la surface
- l est a direction de la lumière
- r est la réflexion de l par rapport à n
- e est la direction de vision
- s est la brillance : c'est un nombre (souvent puissance de deux) pondérant l'étalement de la tâche spéculaire, autrement dit plus ce nombre est élevé et plus la surface paraitra lisse

La figure suivante montre la sigification des paramètres ci-dessus :



Remarque : **n**, **l**, **r**, et **e** étant normalisés, on a :

- $-\mathbf{n}.\mathbf{l} = cos(\theta)$
- $\mathbf{r}.\mathbf{e} = cos(\alpha)$

Shading de Gouraud

Passons maintenant à un vrai calcul d'illumination. Le shading de Gouraud revient à calculer la couleur de chaque sommet d'après le modèle de Phong.

Dans le vertex shader, créez à votre guise les composantes dont vous avez besoin pour le calcul d'illumination (direction de la lumière, direction de la vue, couleurs des différentes composantes, brillance, ...). Calculez ensuite les coefficients angulaires ($max(-\mathbf{n}.\mathbf{l}, 0)$ et $max(\mathbf{r}.\mathbf{e}, 0)^s$). Enfin, combinez ces facteurs pour obtenir la couleur finale du sommet.

Attention, le calcul de la direction de vue demande un peu de réflexion.

Transformation des normales

Jusqu'à maintenant, lorsque nous transformons les positions des sommets, nous ne nous préoccuppons pas de transformer les normales. Ceci n'est pas gênant car nous ne déplaçons pas notre objet (`model_matrix` reste contant). Cependant, si notre objet était modifié, l'absence de transformation des normales créerait un artefact visuel. Ceci vient du fait que les normales sont des *bivecteurs*, et non pas des vecteurs comme les positions. Pour corriger ceci, il faut transformer les normales comme ceci :

```
vert_normal = (transpose(inverse(ModelMatrix)) * vec4(in_normal, 0.0)).xyz;
```

Intuitivement, cette transformation applique la composante rotative et l'inverse de la composante homotétique (scaling) de la matrice du modèle.

Shading de Phong

La gestion de l'éclairage par l'affectation d'une couleur par sommet n'est pas optimale. En effet, la relation entre la couleur et ses composantes n'est pas linéaire. Autrement dit, l'interpolation de deux couleur n'est pas égal à la couleur issue de l'interpolation des composantes. Pour rappel, *OpenGL* interpole linéairement toute les données associées aux sommets pour les affecter aux fragments lors de la rasterisation.

Pour pallier à ce problème, le shading de Phong propose de calculer les composantes de l'illumination en chaque sommet, mais d'effectuer le calcul dans le fragment shader. Ainsi le calcul se fait sur des composantes qui ont étée interpolées.

Pour mettre en place le shading de Phong, vous devez créer des variables de sorties dans votre vertex shader qui seront également des variables d'entrée du fragment shader, comme c'est actuellement le cas pour la variable `vert_color`. Vous pouvez ensuite faire le calcul dans le fragment shader.

Attention : l'interpolation dé-nomalise les vecteurs. Il faut donc re-normaliser la normale dans le fragment shader (à l'aide de `normalize`).

Flat Shading

Comme ous l'avons vu précédemment, les normales des sommets sont interpolées lors de la resterisation. Cependant ceci peut avoir un désavantage : lorsqu'on dessine un cube, on souhaiterait *a priori* voir des arêtes vives, ce qui n'est pas possible avec des normales interpolées.

Pour contrer ce phénomène, nous pourrions dupliquer tout les sommets de manière à ce que chaque sommet ne participe qu'à un seul triangle. Ce faisant, nous perdrions l'avantage de l'indexation.

Pour empêcher *OpenGL* d'interpoler les normales (ou n'importe quel autre attribut de sommet), nous disposons du mot-clef `flat`. Pour l'utiliser, il faut le placer devant la déclaration de l'attribut concerné (devant le `out` dans le vertex shader et devant le `in` dans le fragment shader).

Essayez cette méthode pour visualiser votre maillage.

Autres effets (au choix)

Les parties peuvent être effectuer dans un ordre quelconque.

Orientation de la lumière

Dans le vertex shader, il est possible de modifier la direction de la lumière en fonction du point de vue. Ceci permet par exemple de simuler une lumière provenant de l'observateur.

Essayer de trouver comment réaliser cet effet.

Contrôle de la direction de lumière et des paramètres avec clavier et souris

Essayer de controler la direction de la lumière avec la souris. Il faudra pour cela récupérer la position de la souris, en déduire une orientation dans l'espace, puis l'envoyer à votre shader sous forme de variable uniforme. Vous pouvez faire la même chose (en utilisant le clavier par exemple) pour controler les paramètres de matériaux (comme le shininess).

Couleur et animation

Vous disposez d'une couleur par sommet. En utilisant les fonctions disponibles dans GLSL, utilisez cette information pour la combiner aux rendus que vous obtenez. Objectif : obtenir un rendu plus jolie/original/stylisé/etc. Vous pouvez aussi essayer d'animer les positions des sommets dans le vertex shader à l'aide d'une variable globale (uniforme) que vous pouvez envoyer du CPU vers le GPU (voir TP précédent).

Shading de Blinn-Phong

Un alternative au shading de Phong consiste à modifier le calcul du coefficient spéculaire pour le rendre plus économique. Il s'agit de remplacer $\mathbf{r}.\mathbf{e}$ par $\mathbf{h}.\mathbf{n}$ où :

$$\mathbf{h} = \frac{\mathbf{l} + \mathbf{e}}{\|\mathbf{l} + \mathbf{e}\|}$$

Toon Shading

Le principe du toon shading est d'avoir une image qui ressemble à celle qu'on peut voir dans certain dessins. Ce principe a été mis en place avec succès dans certain jeux (voir XIII par exemple).

Un des principes de base du toon shading consiste à réduire le nombre de couleurs disponible pour l'affichage. Ceci peut être réalisé facilement grâce à un calcul de modulo (cf. fonction `mod`) dans le fragment shader.

Essayer de trouver comment réaliser cet effet, et essayez différent nombre de niveaux de couleur.

Creation d'un maillage

Vous pouvez facilement créer vos propres maillages à vsualiser dans votre programme. Pour ce faire, vous pouvez par exemple utiliser *Blender*. Pour plus d'information sur son installation et son utilisation, voir `http://www.blender.org/` . Une fois votre maillage créé, vous devez l'exporter en OFF. Cette fonctionnalité n'étant pas active par défaut, il faut aller dans `File > User Preferences > Addons > Import-Export: OFF format`. Ensuite, une fois votre objet sélectionné faites `File > Export > OFF Mesh`.

Rendu en *Wire Frame*

Il peut être parfois nécessaire de distinguer clairement les triangles d'un maillage. Pour ce faire, on peut appeller la fonction `glPolygon` avant le dessin.

Essayez. Vous remarquerez que l'absence de face rend la perception de la profondeur difficile. Afin de visualiser les faces et les arêtes, il faut dessiner l'objet deux fois : une fois dans chaque mode. Ceci nécessite également de créer un couple de shaders dédié au rendu des arêtes (sans quoi celles-ci seront de la même couleurs que les faces).

A Position de la caméra

Dans l'équation de l'éclairage, le terme spéculaire $\rho_s.L_s.max(\mathbf{r}.\mathbf{e}, 0)^s$ nécessite de connaitre la direction par laquelle le point éclairé est vu. Cette direction s'exprime aisément par $e = p - c$ où c est la position du point de vue et c est la position du point de la surface.

Comment obtenir la position de la caméra ? Nous ne connaissons pas *c a priori*. Cependant, la matrice de vue V contient cette information : elle représente la transformation allant de l'espace "monde" à l'espace "caméra". Or nous cherchons la position de la caméra dans le monde. Le point étant à l'origine dans l'espace caméra, de coordonnées homogènes $(0, 0, 0, 1)$, est le point de vue dans l'espace "caméra". Pour le faire passer dans l'espace "monde", il faut donc le multiplier par l'inverse de la matrice de vue : V^{-1}.

En GLSL, on peut calculer l'inverse d'une matrice grâce à la fonction `inverse`. On peut donc écrire : `vec4 c_4 = inverse(ViewMatrix)*vec4(0,0,0,1);` Puis, afin de projetter ce point homogène dans l'espace usuel : `vec3 c = c_4.xyz;`

Organisation

Rendre une archive *nom1_ nom2.zip* contenant :

- Le code commenté prêt à être compilé/executé et générant les sorties demandées.
- Un rapport contenant vos résultats (réponses, images, commentaires, ...).

À envoyer par mail à `romain.vergne@inria.fr` avec "[RICM4-TP4] Nom1 Nom2" en objet.