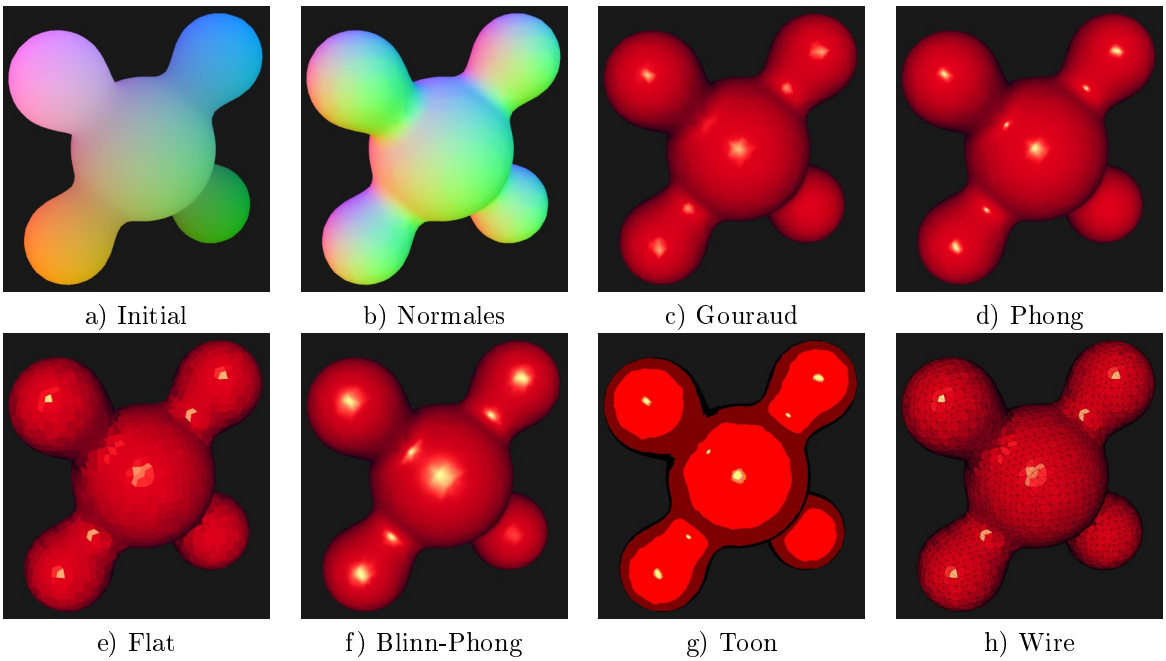
TP4 : Shading



a) Initial b) Normales c) Gouraud d) Phong

e) Flat f) Blinn-Phong g) Toon h) Wire

Figure 1: Les différents shadings proposés dans ce TP

Introduction

L'objectif du TP est de construire un outil de visualisation de modèle 3D simple permettant l'importation et l'affichage de modèle 3D.

La base de code qui vous est fournie reprend les éléments de la dernière séance. Pour la récupérer, allez à l'adresse suivate :

`https://team.inria.fr/imagine/files/2015/01/TP4_01_base_upload.zip` .

Pour l'utiliser, créez un dossier temporaire depuis lequel vous utiliserez CMake pour générer un Makefile avant de lancez la compilation.

```
mkdir build             # Creation du dossier build/
cd build                # Acces au dossier build/
cmake ..                # Creation du Makefile
make                    # Compilation + edition de liens
./polytech_ricm4_tp4    # Execution
```

Contrairement aux TP précédents, les librairies externes et les modèles ne sont pas inclus dans l'archive. Celle-ci contient les adresses auxquelles les récupérer.

Ces dossiers allant être ré-utilisés dans les prochain TP, je vous conseille de les copier une fois pour toute dans votre dossier personnel, et de créer un lien symbolique vers ceux-ci dans le dossier de votre TP.

```
cd MSI/TP4/             # Le dossier MSI contient: external/ models/ TP1/ TP2/ ...
ln -rs ../external/     # Creation d'un lien symbolique vers external/
ln -rs ../models/       # Creation d'un lien symbolique vers models/
```

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
  - ... (ex: BRDF)
- un ensemble de paramètres relatifs à la lumière :
  - une position dans l'espace ou une direction d'illumination
  - une couleur

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 :

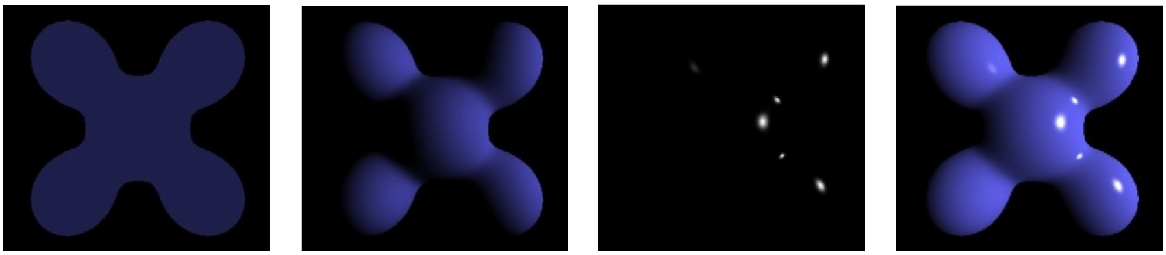


Figure 2: Les différentes composantes du modèle d'illumination de Phong

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 à 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 :

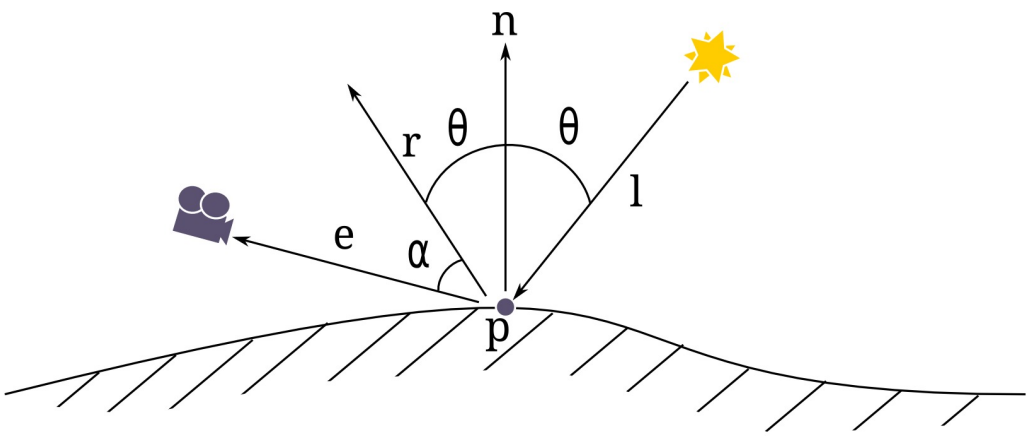


Figure 3: Schéma explicatif des différentes composantse du modèle de Phong.

Transmission des normales

Une première chose à remarque est que le modèle de phong nécessite la normale en un point pour pouvoir calculer sa couleur. La classe `Mesh` contient un attribut `normals` qui donne accès à une normale par sommet. Nous allons passer les normales aux shader par un *VBO*. Vous connaissez maintenant la procédure pour rajouter un *VBO* à votre programme. Voici un rappel. Il faut :

1. créer le *VBO* (`glGenBuffers`)
2. le binder (`glBindBuffer`)
3. y recopier le tableau (`glBufferData`)
4. créer la variable dans le vertex shader (`in`), que vous appelerez `in_normal` par exemple.
5. récupérer l'identifiant de celle-ci dans le main (`glGetAttribLocation`)
6. activer la variable dans la boucle de dessin (`glEnableVertexAttribArray`)
7. binder le buffer (`glBindBuffer`)
8. spécifier son mode de lecture (`glVertexAttribPointer`)

NB : Les étapes 1, 2, 3, et 5 font partie de l'initialisation de `main.cpp`, l'étape 4 se déroule dans `vertex.glsl`, et les étapes 6, 7, et 8 sont dans la boucle de dessin de `main.cpp`.

Réaliser la procédure ci-dessus, et vérifiez son fonctionnement en créant une couleur qui dépend de la normale, par exemple en mettan dan le vertex shader :

```
vert_color = 0.5 * (vec3(1.0) + in_normal)
```

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 ce transformation des normales créerait un artefact visuel. 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

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.

Bonus

Ce qui suit est facultatif. Les parties peuvent être effectuer dans un ordre quelconque, mais sont classés dans l'ordre croissant de difficultée.

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.

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`.

Shading de Blinn-Phong

Un alternative au shading de Phong consiste à modifier le calcul du coefficient spéculaire pour que son calcul soit plus économique. Il s'agit de remplacer **r**.**e** par **h**.**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 disponibl 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.

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 mal aisée. 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).