TP2 - Rendu

1 Calcul de l'ombrage d'un maillage

1. Implémentez la modulation de la couleur ambiante en incrémentant / décrémentant ambientRef (A/a : augmente / diminue la réflection ambiante de 0.1).
2. Dans le fragment shader, ajouter la composante diffuse en calculant l'éclairage de la lumière 0 (gl_LightSource[0]) à I (slide 36). Modulez la en utilisant diffuseRef.
3. Ajouter la composante spéculaire en calculant l'éclairage de la lumière 0 (gl_LightSource[0]) à I (slide 43). Modulez la en utilisant specularRef.
4. Implémentez les intéractions claviers suivantes :
   - D/d : augmente / diminue la réflection diffuse (incrémentant / décrémentant diffuseRef, utilisez un pas de 0.1),
   - S/s : augmente / diminue la réflection spéculaire (incrémentant / décrémentant specularRef, utilisez un pas de 0.1),
   - +/- : augmente / diminue la brillance (incrémentant / décrémentant shininess, utilisez un pas de 1).
5. Itérez sur les 3 lumières définies et ajouter leurs contributions à I.

1. Modifions la fonction key en ajoutant le code suivant :

```
void key (unsigned char keyPressed, int x, int y) {
    switch (keyPressed) {
    // ...
    case 'A':
        ambientRef = min(1.f, ambientRef + 0.1f);
        break;
    case 'a':
        ambientRef = max(0.f, ambientRef - 0.1f);
        break;
    }
}
```

2. On cherche à calculer I_d l'intensité de la lumière diffuse réfléchie. D'après la loi de Lambert, on a :

$$I_d = I_{sd} * K_d * cos\theta$$

avec :
- I_{sd} l'intensité de la lumière diffuse (gl_LightSource[0].diffuse),
- $K_d \in [0, 1]$ le coefficient de réflexion diffuse du matériau (gl_FrontMaterial.diffuse),
- θ l'angle entre le vecteur (point à éclairer - source de lumière) et la normale de ce point.

Soit $\vec{L}$ le vecteur entre la source de lumière et le point à éclairer. Soit $\vec{N}$ la normale du point. On peut aussi écrire :

$$I_d = I_{sd} * K_d * (\vec{L}.\vec{N})$$

si et seulement si $\vec{L}$ et $\vec{N}$ sont normalisés (et que le produit scalaire des deux vecteurs est positif). Ainsi, pour calculer la diffuse, dans le fragment du shader (shader.frag), on doit ajouter ce bout de code :

```
// ...
uniform float diffuseRef;

varying vec4 p;
varying vec3 n;

void main (void) {
    // Position P du point à éclairer, et normale N en ce point.
    vec3 P = vec3(gl_ModelViewMatrix * p);
    vec3 N = normalize(gl_NormalMatrix * n);

    // Calcul de l'ambiante, I est déclaré...

    vec4 Isd = gl_LightSource[i].diffuse;
    vec4 Kd  = gl_FrontMaterial.diffuse;
    vec3 L   = normalize(gl_LightSource[0].xyz - P);
    vec4 Id  = Isd * Kd * max(dot(N, L), 0.);
    I += Id * diffuseRef

    // ...
}
```

3. On cherche maintenant à calculer I_s l'intensité de la lumière spéculaire réfléchie. D'après le modèle de Phong, on a :

$$I_s = I_{ss} * K_s * cos\alpha^b$$

avec :

- I_{ss} l'intensité de la lumière spéculaire (gl_LightSource[0].specular),
- $K_s \in [0, 1]$ le coefficient de réflexion spéculaire du matériau (gl_FrontMaterial.specular),
- α l'angle entre les directions de réflexion et de la vue,
- b la brillance.

Soit $\vec{V}$ le vecteur de vue. Soit $\vec{R}$ le vecteur de réflexion de la lumière sur un miroir. On a :

$$\vec{R} = 2(\vec{N}.\vec{L})\vec{N} - \vec{L}$$

Avec ces éléments, le modèle de Phong pour calculer la spéculaire peut aussi s'écrire avec l'équation suivante :

$$I_s = I_{ss} * K_s * (\vec{R}.\vec{V})^n$$

Ici, normalement, $\vec{R}$ et $\vec{V}$ sont déjà normalisés, donc le résultat est aussi normalisé. Ainsi, pour calculer la spéculaire, le bout de code suivant est nécessaire :

```
void main(void) {
    // ...
    vec3 V = normalize(-P);
    // ...
    vec4 Iss = gl_LightSource[0].specular;
    vec4 Ks  = gl_FrontMaterial.specular;
    // Il n'y a pas d'entier en GLSL, on est obligés de mettre 2. pour que ça marche.
```

```
    vec3 R   = (2. * dot(N, L) * N) - L;
    vec4 Is  = Iss * Ks * pow(max(dot(R, V), 0.), shininess);
    I += Is * specularRef;
    // ...
}
```

4. Comme dans la première question, on ajoute juste des cas au switch de key :

```
void key (unsigned char keyPressed, int x, int y) {
    switch (keyPressed) {
    // ...
    case 'D' :
        diffuseRef = min(1.f, diffuseRef + 0.1f);
        break;
    case 'd' :
        diffuseRef = max(0.f, diffuseRef - 0.1f);
        break;
    case 'S' :
        specularRef = min(1.f, specularRef + 0.1f);
        break;
    case 's' :
        specularRef = max(0.f, specularRef - 0.1f);
        break;
    case '+' :
        shininess += 1.f;
        break;
    case '-' :
        shininess = max(0.f, shininess - 1.f);
        break;
    // ...
    }
}
```

5. Il suffit d'ajouter une boucle autour du calcul de la diffuse et de la spéculaire. Voici le fichier final shader.frag obtenu :

```
uniform float ambientRef;
uniform float diffuseRef;
uniform float specularRef;
uniform float shininess;

varying vec4 p;
varying vec3 n;

void main (void) {
    vec3 P = vec3(gl_ModelViewMatrix * p);
    vec3 N = normalize(gl_NormalMatrix * n);
    vec3 V = normalize(-P); //Vecteur de vue

    // Ambiante
    vec4 Isa = gl_LightModel.ambient;
    vec4 Ka  = gl_FrontMaterial.ambient;
    vec4 Ia  = Isa * Ka;
    vec4 I = ambientRef * Ia;
```

```
    for (int i = 0; i < 3; ++i) {
        // Diffuse
        vec4 Isd = gl_LightSource[i].diffuse;
        vec4 Kd  = gl_FrontMaterial.diffuse;
        vec3 L   = normalize(gl_LightSource[i].position.xyz - P);
        vec4 Id  = Isd * Kd * max(dot(N, L), 0.);
        I += Id * diffuseRef;

        // Spéculaire
        vec4 Iss = gl_LightSource[i].specular;
        vec4 Ks  = gl_FrontMaterial.specular;
        vec3 R   = (2. * dot(N, L) * N) - L;
        vec4 Is  = Iss * Ks * pow(max(dot(R, V), 0.), shininess);
        I += Is * specularRef;
    }

    gl_FragColor = vec4 (I.xyz, 1);
}
```