TP N°10
L'Éclairement

1- Lumière ambiante :

Avant d'utiliser une source lumineuse, on applique un éclairage ambiant et on s'assure de pouvoir régler son intensité :

- On définit une couleur de lumière avec une variable uniforme :

```
vec3  lightColor(1.0f, 1.0f, 1.0f) ;
GLuint LightClrID  = glGetUniformLocation(ShaderProgram, "lightColor");
glUniform3fv(LightClrID, 1, &lightColor[0]);
```

- Dans le fragment shader, multiplier la couleur de l'objet par la couleur de la lumière :

```
in vec3 vColor;
out vec4 color;

uniform vec3 lightColor;

void main()

{

   color = vec4(lightColor * vColor, 1.0);

}
```

On peut ajouter un paramètre pour régler l'intensité de la lumière ambiante :

```
float ambientIntens = 0.1;
vec3 ambient= lightColor * ambientIntens;
color = vec4( ambient * vColor, 1.0 );
```

2- Lumière diffuse :

Nous allons à présent, ajouter une source lumineuse. Pour calculer l'intensité de la réflexion diffuse d'un fragment, nous avons besoins du vecteur normal et de la direction de la source lumineuse :

- On récupère les normales à partir du fichier obj, et on les envoie aux buffers:

```
glGenBuffers(1, &VBONormal);
glBindBuffer(GL_ARRAY_BUFFER, VBONormal);
glBufferData(GL_ARRAY_BUFFER, normals.size() * sizeof(vec3), &normals[0], GL_STATIC_DRAW);
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE,3 * sizeof(GLfloat), (void*)0 );
```

- Le vertex shader récupère les normales :

```
layout(location = 1) in vec3 VertexNormal;
out vec3 vNormal;
```

et les renvoie au fragment shader :

```
vNormal =  VertexNormal ;
```

- Utiliser une variable uniforme pour fixer la position de la source lumineuse :

```
vec3 lightPos(1.0f, 1.0f, 5.0f);
GLuint LightPosID  = glGetUniformLocation(ShaderProgram, "lightPos");
glUniform3fv(LightPosID, 1, &lightPos[0]);
```

- Nous avons besoin de calculer l'éclairement dans le repère du monde réel, nous utilisons, donc, la position des fragments selon la matrice *Model* uniquement. Il faut alors récupérer la matrice *Model* via une variable uniforme. Puis, dans le vertex shader, la position sera calculée :

```
out vec3 vPos;
uniform mat4 Model;
```

calcul et envoie au fragment shader :

```
vPos = Model * vec4(vertexPosition, 1);
```

- Nous pouvons, à présent, calculer la réflexion diffuse avec le produit scalaire du vecteur normal et le vecteur direction de la lumière. On n'oublie pas de normaliser les deux vecteurs :

```
vec3 normal = normalize(vNormal);
vec3 lightDir = normalize(lightPos - vPos);
float diff = max(dot(normal, lightDir), 0.0);
vec3 diffuse = diff * lightColor;
```

Vous remarquez que si le produit scalaire (normal . LightDir) < 0 alors l'éclairage est nul.
Une variable pour l'intensité de la lumière peut être définie, puis multipliée par le résultat.

3- Lumière spéculaire :

Afin d'avoir un effet de réflexion spéculaire, nous avons besoin du vecteur direction de la visualisation en plus des vecteurs de direction de la lumière et du vecteur normal.
Il faut récupérer la position de la caméra avec une variable uniforme, puis calculer l'intensité de la réflexion spéculaire (telles que vue en cours) :

```
float specIntens = 0.5;
vec3 viewDir = normalize(viewPos - vPos);
vec3 reflectDir = reflect(-lightDir, normal);
float spec = pow(max(dot(viewDir, reflectDir), 0.0),  50);
vec3 specular = specIntens * spec * lightColor;
```

On fait la somme des trois intensités, ambiante, diffuse et spéculaire :

```
color = vec4((ambient + diffuse + specular) * vColor, 1.0);
```

On obtient le rendu suivant :