INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

Informatique Graphique - TP 6

Illumination locale

GRAINDORGE Amance

VERNET Hector

Université de Rennes

I. INTRODUCTION

Dans ce TP, nous allons implémenter le modèle de Phong pour illuminer notre scène dynamiquement. Ce modèle comprend des composantes ambiante, diffuse et spéculaire, qui permettent d'approximer simplement l'interaction de la lumière avec les surfaces, tout en exposant beaucoup de paramètres nous donnant un contrôle artistique sur le résultat. Ce modèle sera implémenté pour différentes sources lumineuses : directionnelle, ponctuelle et spot.

II. EXERCICE 1

Cet exercice consiste simplement à vérifier que l'implémentation déjà présente fonctionne correctement. Dans `phongFragment.glsl`, pour le calcul de la lumière directionnelle (une lumière infiniment distante aux rayons parallèles, le cas le plus simple à traiter), on retrouve les trois composantes du modèle de Phong :

- La composante ambiante, dépendant simplement de deux constantes :

```
1    vec3 ambient = light.ambient * material.ambient;
```

- La composante diffuse, facteur du produit scalaire entre la normale et la direction de la lumière :

```
1    float diffuse_factor = max(dot(surfel_normal, surfel_to_light), 0.0);
2    vec3 diffuse = diffuse_factor * light.diffuse * material.diffuse ;
```

- La composante spéculaire,

```
1    vec3 reflect_direction = reflect(-surfel_to_light, surfel_normal);
2    float specular_dot = clamp(dot(surfel_to_camera, reflect_direction), 0, 1);
3    float specular_factor = pow(specular_dot, material.shininess);
4    vec3 specular = specular_factor * light.specular * material.specular;
```

La couleur finale est obtenue en faisant la somme de ces trois composantes.

III. EXERCICE 2

Dans cet exercice, nous allons ajouter le concept d'atténuation de la lumière, qui fait qu'elle diminue de manière proportionnelle au carré de la distance parcourue. Cette atténuation dépend également de trois coefficients propres (constant, linéaire et quadratique) qui permettent de contrôler la rapidité de cette diminution.

La formule est la suivante :

```
1    float attenuation = 1.0 / (light.constant + light.linear * distance + light.quadratic *
     (distance * distance));
```

On l'implémente pour les lumières ponctuelles et spot uniquement, étant donné que les lumières directionnelles sont supposées infiniment éloignées et uniformes sur toute la scène (il s'agira typiquement du soleil).

Voici un avant/après de l'effet d'atténuation avec la scène d'example :

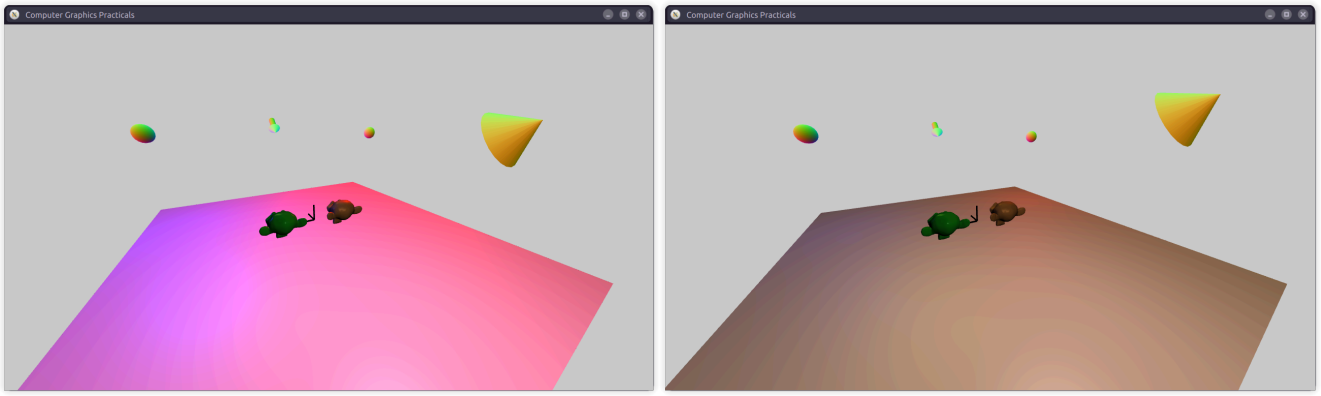

Figure 1 : Effet de l'atténuation sur la scène d'exemple

IV. EXERCICE 3

Dans cet exercice, nous allons implémenter l'« ouverture » des lumières spot, qui permet de définir un cône d'illumination. Dans un premier temps, nous n'utiliserons q'un seul seuil, ce qui donnera une coupure nette de la lumière en dehors du cône.

La formule est donc celle-ci (`light.innerCutOff` étant l'angle d'ouverture, ou le seuil) :

```
1    float intensity = max(dot(surfel_to_light, -light.spotDirection), 0.0) <
     cos(light.innerCutOff / 2) ? 0.0 : 1.0;
```

Cette intensité est ensuite multipliée à la lumière diffuse et spéculaire.

Voici le spot formé :

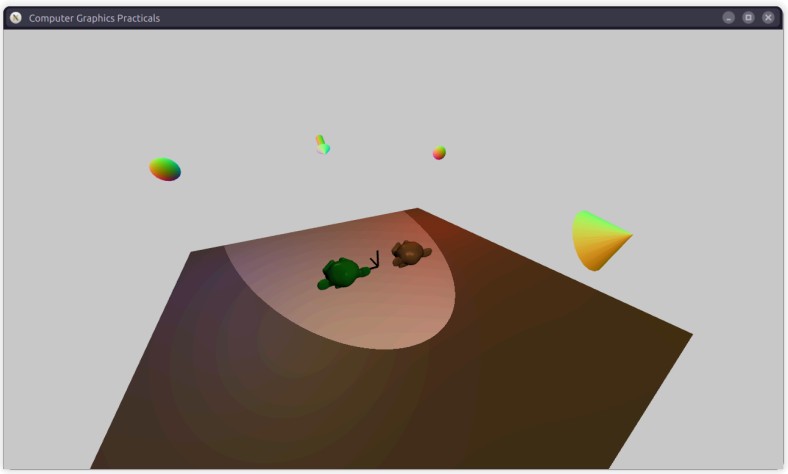

Figure 2 : Lumière spot avec un seul seuil

V. EXERCICE 4

À présent, nous allons ajouter un deuxième seuil pour adoucir la coupure de la lumière spot. Nous utilisons pour cela deux angles : un angle interne « `innerCutOff` » et un angle externe « `outerCutOff` ».

Voici la formule de l'intensité révisée :

```
1    float cos_phi = max(dot(surfel_to_light, -light.spotDirection), 0.0);
2    float intensity = clamp((cos_phi - cos(light.outerCutOff / 2))/(cos(light.outerCutOff /
     2) - cos(light.innerCutOff / 2)),0,1);
```

Avec cette nouvelle formule, on obtient une transition plus douce :

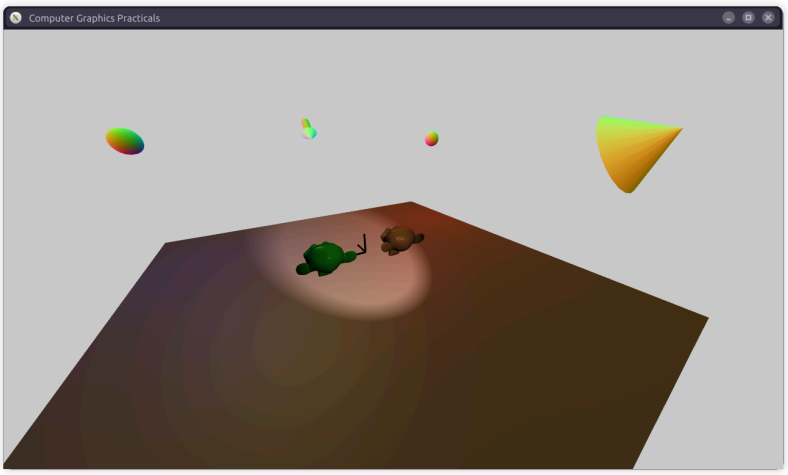

Figure 3 : Lumière spot avec transition douce

VI. EXERCICE 5

Dans cet exercice, nous allons profiter du fait que les lumières héritent de la classe `KeyframedHierarchicalRenderable` pour les animer dans notre scène de test.

On ajoute un source lumineuse en mouvement dans `practical6.cpp` :

```
p_position = glm::vec3(0.0, 0.0, 0.0);
p_ambient = glm::vec3(0.0, 0.0, 0.0);
p_diffuse = glm::vec3(2.0, 2.0, 2.0);
p_specular = glm::vec3(2.0, 2.0, 2.0);
p_constant = 1.0;
p_linear = 5e-1;
p_quadratic = 0;
PointLightPtr pointLight3 = std::make_shared<PointLight>(p_position, p_ambient,
p_diffuse, p_specular, p_constant, p_linear, p_quadratic);
pointLight3->addGlobalTransformKeyframe(
  getTranslationMatrix(glm::vec3(5.0, 0.0, 2.0)), 0.0);
pointLight3->addGlobalTransformKeyframe(
  getTranslationMatrix(glm::vec3(-5.0, 0.0, 2.0)), 1.0);
pointLight3->addGlobalTransformKeyframe(
  getTranslationMatrix(glm::vec3(5.0, 0.0, 2.0)), 2.0);
pointLight3->addGlobalTransformKeyframe(
  getTranslationMatrix(glm::vec3(-5.0, 0.0, 2.0)), 3.0);
viewer.addPointLight(pointLight3);
```

En lançant la scène, on voit la lumière faire des aller-retours devant les Suzannes :

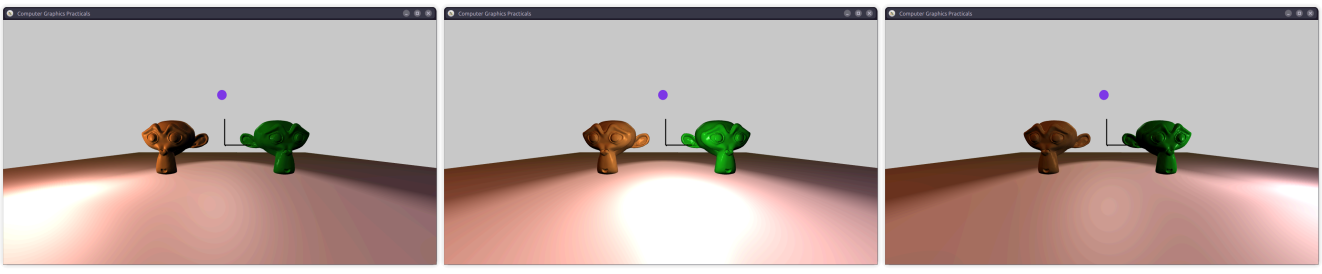

Figure 4 : Lumière ponctuelle animée

VII. EXERCICE 6

Dans cet exercice, nous allons adapter le shader `cartoonFragment.glsl` pour fonctionner avec des textures et un nombre arbitraire de lumières. On écrit donc `cartoonTextureFragment.glsl` :

```
// [...] Similaire à textureFragment.glsl

float posterizeFactor(float value, float steps) {
    return floor(value * steps) / steps;
}

void main()
{
    //Surface to camera vector
    vec3 surfel_to_camera = normalize( cameraPosition - surfel_position );

    int clampedNumberOfDirectionalLight =
      max(0, min(numberOfDirectionalLight, MAX_NR_DIRECTIONAL_LIGHTS));
    int clampedNumberOfPointLight =
      max(0, min(numberOfPointLight, MAX_NR_POINT_LIGHTS));
    int clampedNumberOfSpotLight =
      max(0, min(numberOfSpotLight, MAX_NR_SPOT_LIGHTS));

    vec3 tmpColor = vec3(0.0, 0.0, 0.0);

    for(int i=0; i<clampedNumberOfDirectionalLight; ++i)
        tmpColor += computeDirectionalLight(directionalLight[i], surfel_to_camera);

    for(int i=0; i<clampedNumberOfPointLight; ++i)
        tmpColor += computePointLight(pointLight[i], surfel_to_camera);

    for(int i=0; i<clampedNumberOfSpotLight; ++i)
        tmpColor += computeSpotLight(spotLight[i], surfel_to_camera);

    vec4 textureColor = texture(texSampler, surfel_texCoord);

    if(textureColor.a < 0.8)
      discard;

    textureColor = textureColor * posterizeFactor(sqrt(length(textureColor)), 8.0);

    outColor = textureColor*vec4(tmpColor,1.0);

}
```

En réalité, l'effet n'est plus vraiment « cartoon » car la postérisation a beaucoup d'étapes (8), mais cela permet d'appliquer le shader à toute la scène sans trop sacrifier de qualité visuelle avec des grands zones sombres. Le résultat est tout de même assez stylisé et donne un look retro (ce qui n'est de tout façon pas évitable à cause de la simplicité du modèle d'éclairage, donc autant en jouer) :

Figure 5 : Titre de l'animation postérisé

En terme d'autres améliorations, nous aimerions aussi ajouter un effet de contour noir autour des objets, et un effet de brouillard lorsque la caméra est sous l'eau (avec un shader de post processing, SFML le permet assez facilement normalement).

VIII. CONCLUSION

En conclusion, ce TP nous a permis d'implémenter le modèle d'illumination de Phong avec différentes sources lumineuses et des effets d'atténuation et de spot. Nous avons également adapté le shader cartoon pour fonctionner de manière plus universelle, ce qui donne rendu stylisé à notre scène.