Rapport Tp2 ISIR

jeremy Vacher

February 2024

Table des matières

1 Introduction

Dans ce Tp, nous allons comprendre le fonction d'une autre source lumineuse et faire le calcul d'ombres portées en lançant des rayon d'ombrages permettant de savoir si le point est visible depuis la source lumineuse ou invisible.

2 L'objet plan

Dans ce Tp, nous devons commencer par créer une classe nous permettant de rajouter un plan dans notre scène. Notre classe sera composé de deux variables qui représente un point sur le plan et la normale du plan. Nous aurons aussi une fonction **intersect** nous permettant de vérifier l'intersection avec le plan.

Dans notre fonction, on va vérifier l'intersection entre le plan et les rayons. Pour cela, on calcule le dénominateur qui va être le produit scalaire entre la normale du plan et la direction de notre rayon. Si le dénominateur est différent de zéro, ça implique que le rayon et le plan ne sont pas parallèles. Ensuite, on vérifie si l'intersection se trouve devant le point d'origine du rayon ou derrière.

```
bool PlaneGeometry::intersect( const Ray &p_ray, float & p_t1 )
const {
    float den = glm::dot( _normaleP, p_ray.getDirection() );
    if (den != 0) {
        Vec3f po = _positionP - p_ray.getOrigin();
        p_t1     = glm::dot( po, _normaleP ) / den;
        return p_t1 > 0 ? true : false;
    }
    return false;
}
```

Enfin dans la scène, on rajoute un plan de couleur rouge que l'on associe à notre matériaux. Nous obtenons alors cette image :

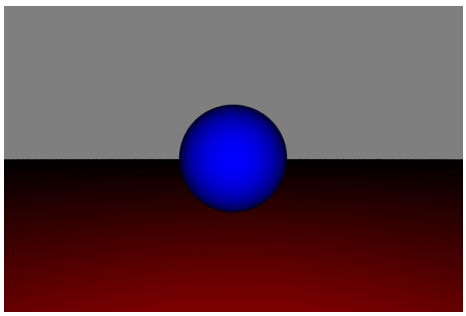

FIGURE 1 – Résultat exercice 1

3 Sources lumineuses ponctuelles

Maintenant, nous devons rajouter une classe nous permettant de définir une lumière ponctuelle. Pour cela, nous avons une variables représentant la position

de la lumière. Ensuite comme la classe hérite de **baseLight**, on rajoute dans notre constructeur les paramètres de **baseLight**.

```
PointLight( const Vec3f & p_position, const Vec3f & p_color,
const float p_power = 1.f )
: _position( p_position ), BaseLight(p_color,p_power)
{
}
```

On ajoute maintenant une fonction **sample** qui retournera à partir d'une position un objet de type **LightSample**. On commence donc par calculer la distance entre le point d'éclairage et la source lumineuse en utilisant la méthode **distance** d'openGL. Ensuite on calcule la radiance qui correspond à la couleur de notre lumière multipliée par sa puissance que l'on divise par notre distance au carré. Puisque nous retournons un objet de type **LightSample**, on doit aussi calculer la direction qui correspond à la position de notre source lumineuse moins le point que l'on souhaite éclairer. Enfin, on rajoute une variable **pdf** égale à 1 et on retourne notre objet **LightSample**. Nous avons alors une fonction comme ceci :

```
LightSample PointLight::sample( const Vec3f & p_point ) const
{
    float _distance = glm::distance( _position, p_point );
    Vec3f _radiance     = ( _color * _power ) / (_distance * _distance);
    Vec3f _direction = glm::normalize( _position - p_point );
    float _pdf              = 1.f;
    return LightSample( _direction, _distance, _radiance, _pdf);
}
```

4 Prise en compte des lumières

Maintenant nous allons rajouter une classe nous permettant de calculer l'éclairage en fonction des sources lumineuses. Nous allons ajouter dans cette classe une méthode **Li** permettant de calculer l'éclairage des objets et une méthode **directLighting** pour calculer l'éclairage privé.

Dans mon programme la méthode Li va utiliser la méthode **directLighting** pour faire l'éclairage si nous avons une intersection. Dans la méthode **directLighting**, on commence par faire une boucle sur la lumière de notre scène. Ensuite on calcule l'angle entre la direction de notre lumière et la normale au point d'intersection. Puis on calcule la luminance finale.

```
Vec3f colorF = VEC3F_ZERO;//luminance finale
for (BaseLight* light : p_scene.getLights()) {
    LightSample ls = light->sample( p_hitRecord._point );
    float theta    = glm::dot( p_hitRecord._normal,ls._direction );
    float maxAngle = glm::max( theta, 0.f );
    colorF += p_hitRecord._object->getMaterial()->
    getFlatColor() * ls._radiance * maxAngle;
}
return colorF;
```

Ensuite on rajoute une scène avec une source lumineuse positionnée en (1, 10, 1) de couleur blanche et de puissance 100. On obtiens alors l'image suivante :

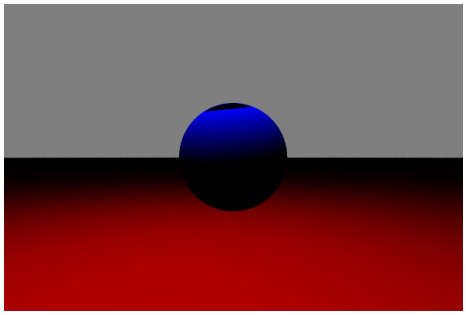

FIGURE 2 – Résultat exercice 3a

Pour régler le problème avec l'image obtenue précédemment on rajoute la méthode clamp dans notre méthode **renderImage**. Cette méthode va nous permettre d'avoir une valeur de couleur entre 0 et 1. On obtiens alors l'image ci-dessous.

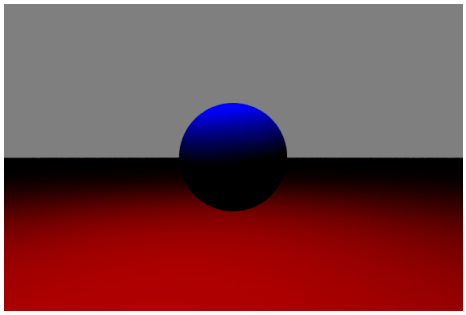

FIGURE 3 – Résultat exercice 3b

5 Calcul d'ombre portées

Dans cette partie nous voulons rajouter des ombres à notre image. Pour cela, nous allons rajouter dans la méthode **directLighting** des rayons d'ombrages. On va donc créer un rayon d'ombrage et on va regarder l'intersection entre notre rayon et le point d'intersection. Enfin, si nous n'avons pas d'ombre on calcule la lumière de la scène avec l'objet.

Pour régler le problème de l'image on utilise la méthode offset qui va décaler l'origine du rayon d'ombrage selon la normale. On obtiens alors l'image ci-dessous.

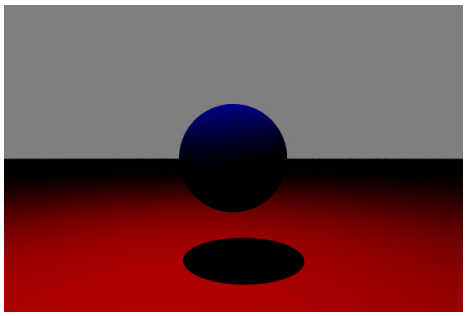

FIGURE 4 – Résultat exercice 4b

6 Recherche d'intersection pour les rayons d'ombrage

Pour cette dernière partie on remplace simplement la méthode **intersect** dans la méthode **directLighting** par la méthode **intersectAny**. On obtiens alors l'image finale suivante.

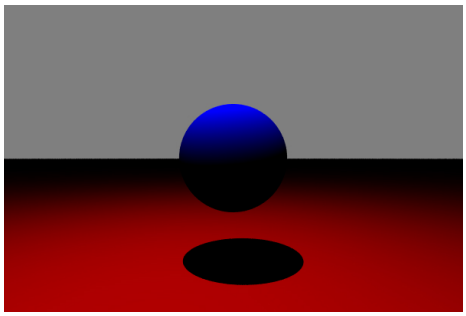

FIGURE 5 – Résultat exercice 5