Université de Limoges
Faculté des Sciences & Techniques
Faculté des Sciences et Techniques de Limoges

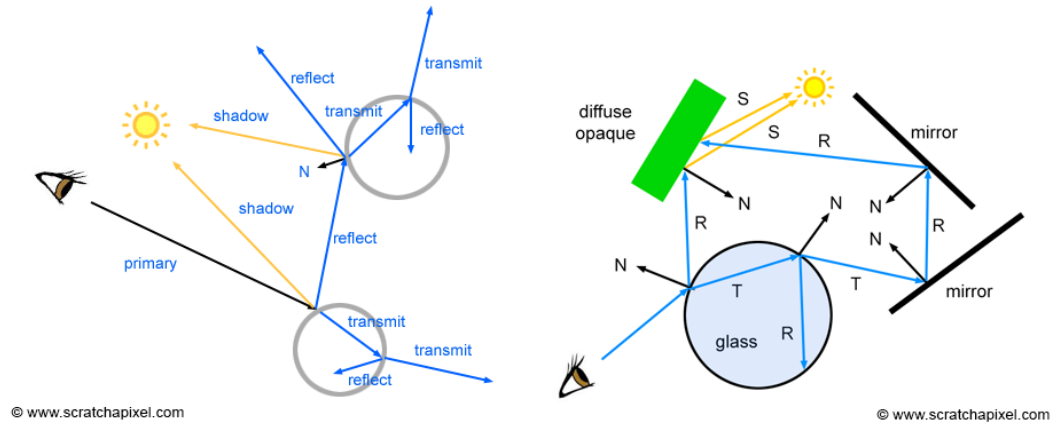
ISIR - TP6 - Réflexions, réfractions

Maxime MARIA

2022-2023

Dans ce TP, nous allons ajouter les phénomènes lumineux de réflexion et de réfraction. Nous allons rester dans le contexte simple de la méthode de Whitted [Whi80] qui a introduit le lancer de rayons en tant que processus récursif : selon le matériau intersecté, le rayon peut être réfléchi et/ou réfracté (ou transmis). La Figure 1 illustre ce principe. En pseudo-code :

```
trace(scene, rayon)
    if <intersection rayon/scene>
        if <materiau mirroir>
            rayonReflexion = <calculer reflexion>
            trace(scene, rayonReflexion)
        else if <materiau transparent>
            rayonReflexion = <calculer reflexion>
            trace(scene, rayonReflexion)
            rayonRefraction = <calculer refraction>
            trace(scene, rayonRefraction)
        else
            <calculer eclairage direct>
```

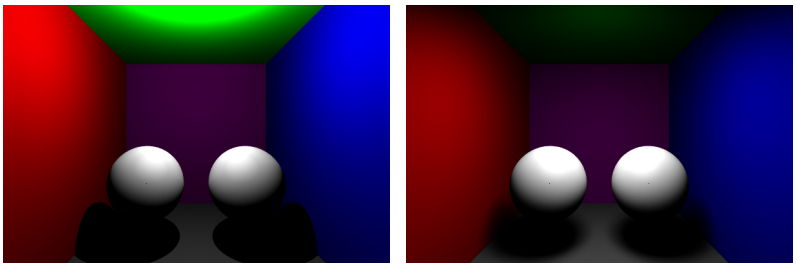


(a) (b)

Figure 1 – Illustration d'un lancer de rayons récursif.

Configuration de la scène

1. Utilisez la scène donnée en Annexe A (à mettre dans `Scene::init()`).
2. Placez la caméra en $(0, 2, -6)$ et orientez-la pour qu'elle regarde dans la direction de l'axe z.
3. Lancez le calcul de l'image, vous devriez obtenir l'image 2(a). En utilisant, la lumière surfacique (en commentaire) à la place de la lumière ponctuelle, vous devriez obtenir l'image 2(b).



(a) (b)

FIGURE 2 – Images de base.

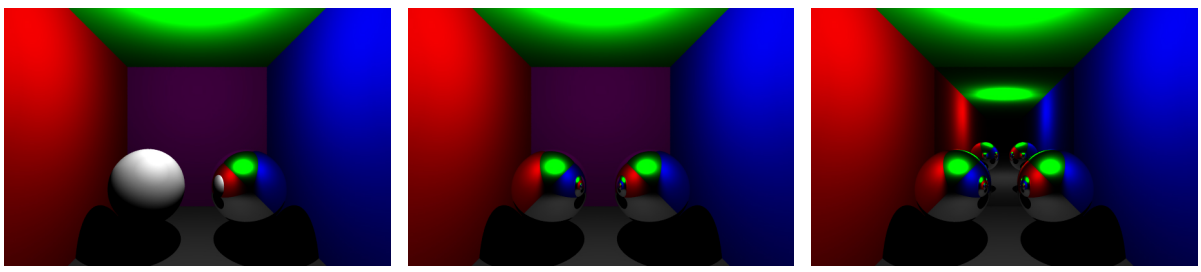
1 Nouvel intégrateur

Ici, nous allons créer un nouvel intégrateur permettant de gérer la récursivité du lancer de rayons.

1. Créez un nouvel intégrateur (`WhittedIntegrator`) à partir de `DirectLightingIntegrator`.
2. Comme tout algorithme récursif, il faut s'assurer d'éviter une boucle d'appels infinie (imaginez deux miroirs face à face). Pour cela, il faut déterminer un critère de sortie. Ici, nous allons simplement compter le nombre de rebonds du rayon. Si celui-ci dépasse un seuil donné, alors la récursion s'arrête et retourne la couleur noire. Ajoutez un attribut `_nbBounces` qui représente le nombre de rebonds maximal qu'un rayon peut effectuer. Par défaut, le nombre de rebonds est fixée à 5.
3. Nous reviendrons sur cette classe dans la suite du TP.

2 Matériau parfaitement spéculaire : le miroir

1. Ajoutez un nouveau matériau (`MirrorMaterial`) qui représentera les objets parfaitement spéculaires. Ce matériau n'a pas d'attribut en particulier. Ses méthodes `shade` et `getColor` retournent simplement la couleur noire (`BLACK`).
2. Dans l'intégrateur, il va falloir identifier si un matériau est un mirroir ou non. Pour cela, ajoutez une méthode `isMirror` à votre classe `BaseMaterial` qui retourne `false`. Surchargez ensuite cette méthode dans `MirrorMaterial` pour qu'elle retourne `true`.
3. Modifiez `WhittedIntegrator` pour prendre en compte les réflexions miroirs dans la scène. La fonction `Li` doit maintenant lancer une fonction récursive (pensez au critère d'arrêt) ! Vous pouvez utiliser `glm::reflect` pour calculer la direction du rayon réfléchi (symétrique par rapport à la normale).
4. En appliquant le matériau miroir à `Sphere1` vous devriez obtenir l'image 3(a). En appliquant un miroir sur `Sphere2` vous devriez obtenir l'image 3(b). Enfin, en appliquant un miroir sur `PlaneFront` vous devriez obtenir l'image 3(c).



(a) (b) (c)

FIGURE 3 – Résultats de l'exercice 2.

3 Matériau transparent : Fresnel

1. Ajoutez un nouveau matériau (`TransparentMaterial`) qui représentera les objets transparents. Ce matériau possède un indice de réfraction (`_ior`) qui est un simple scalaire (nous ne prendrons pas en compte les indices de réflexions complexes). Par défaut, cet attribut vaut 1.3. Ses méthodes `shade` et `getColor` retournent la couleur noire (`BLACK`).
2. Dans l'intégrateur, il va falloir identifier si un matériau est transparent ou non. Pour cela, ajoutez une méthode `isTransparent` à votre classe `BaseMaterial` qui retourne `false`. Surchargez ensuite cette méthode dans `TransparentMaterial` pour qu'elle retourne `true`.
3. Il va aussi falloir pouvoir récupérer l'indice de réfraction dans l'intégrateur. Pour cela, ajoutez une méthode `getIOR` à votre classe `BaseMaterial` qui retourne `1`. Surchargez ensuite cette méthode dans `TransparentMaterial` pour qu'elle retourne `_ior` (On commence à toucher les limites de l'architecture du moteur telle quelle... :-p).
4. Modifiez `WhittedIntegrator` pour prendre en compte les objets transparents dans la scène :
   — L'équation de Fresnel [1] détermine la proportion de lumière réfléchie (k_r) à partir de la direction d'incidence, la normale à la surface et les indices de réfraction. Selon la loi de conservation de l'énergie, la proportion de lumière transmise (réfractée) est donc $1 - k_r$. Pensez à gérer le cas de la réflexion totale !
   — La direction de réfraction est donnée par les lois de Snell-Descartes [2]. Vous pouvez utiliser `glm::refract` pour la calculer ;

   Nous partons du principe que l'origine des rayons primaires se situe dans le vide (indice de réfraction = 1). Attention, il faut prendre en compte le fait que l'on puisse entrer ou sortir d'un objet transparent (utilisez un paramètre supplémentaire dans votre fonction récursive).
5. En appliquant un matériau miroir à `Sphere1` et transparent à `Sphere2`, vous devriez obtenir l'image 4.

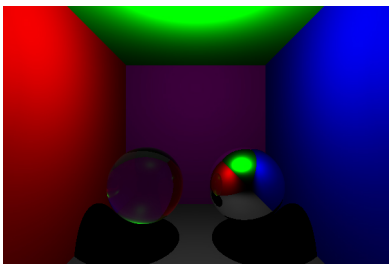


FIGURE 4 – Résultats de l'exercice 3.

N.B. Nous ne tiendrons pas compte de la réfraction pour les rayons d'ombrages. C'est un peu plus compliqué, il pourrait y avoir des caustiques.

1. https://en.wikipedia.org/wiki/Fresnel_equations
2. https://en.wikipedia.org/wiki/Snell%27s_law

A Configuration de la scène

```
// ====================================================================
// Add materials.
// ====================================================================
_addMaterial( new MatteMaterial( "WhiteMatte", WHITE, 0.6f ) );
_addMaterial( new MatteMaterial( "RedMatte", RED, 0.6f ) );
_addMaterial( new MatteMaterial( "GreenMatte", GREEN, 0.6f ) );
_addMaterial( new MatteMaterial( "BlueMatte", BLUE, 0.6f ) );
_addMaterial( new MatteMaterial( "GreyMatte", GREY, 0.6f ) );
_addMaterial( new MatteMaterial( "MagentaMatte", MAGENTA, 0.6f ) );

// ====================================================================
// Add objects.
// ====================================================================
// Spheres.
_addObject( new Sphere( "Sphere1", Vec3f( -2.f, 0.f, 3.f ), 1.5f ) );
_attachMaterialToObject( "WhiteMatte", "Sphere1" );
_addObject( new Sphere( "Sphere2", Vec3f( 2.f, 0.f, 3.f ), 1.5f ) );
_attachMaterialToObject( "WhiteMatte", "Sphere2" );
// Pseudo Cornell box made with infinite planes.
_addObject( new Plane( "PlaneGround", Vec3f( 0.f, -3.f, 0.f ),
                                      Vec3f( 0.f, 1.f, 0.f ) ) );
_attachMaterialToObject( "GreyMatte", "PlaneGround" );
_addObject( new Plane( "PlaneLeft", Vec3f( 5.f, 0.f, 0.f ),
                                    Vec3f( -1.f, 0.f, 0.f ) ) );
_attachMaterialToObject( "RedMatte", "PlaneLeft" );
_addObject( new Plane( "PlaneCeiling", Vec3f( 0.f, 7.f, 0.f ),
                                       Vec3f( 0.f, -1.f, 0.f ) ) );
_attachMaterialToObject( "GreenMatte", "PlaneCeiling" );
_addObject( new Plane( "PlaneRight", Vec3f( -5.f, 0.f, 0.f ),
                                     Vec3f( 1.f, 0.f, 0.f ) ) );
_attachMaterialToObject( "BlueMatte", "PlaneRight" );
_addObject( new Plane( "PlaneFront", Vec3f( 0.f, 0.f, 10.f ),
                                     Vec3f( 0.f, 0.f, -1.f ) ) );
_attachMaterialToObject( "MagentaMatte", "PlaneFront" );

// ====================================================================
// Add lights.
// ====================================================================
_addLight( new PointLight( Vec3f( 0.f, 5.f, 0.f ), WHITE, 100.f ) );
//_addLight( new QuadLight( Vec3f( 1.f, 5.f, -2.f ),
//                          Vec3f( -2.f, 0.f, 0.f ),
//                          Vec3f( 0.f, 1.f, 2.f ), WHITE, 40.f ) );
```

Références

[Whi80] Turner Whitted. An improved illumination model for shaded display. *Commununcations of the ACM*, 23(6) :343–349, June 1980.