MOS 2.2 - Informatique Graphique
Rapport

GOBERT Grégoire

Mars 2020

Table des matières

1 Introduction **1**

2 Scène **2**
2.1 Objets 2
2.2 Sources de lumière 2
2.3 Caméra 3

3 Interaction lumière matière **5**
3.1 Matériau parfaitement diffus 5
3.2 Matériau parfaitement spéculaire 5
3.3 Matériau diffus et spéculaire : modèle de Phong 6
3.4 Matériau transparent 6
3.5 Pixel et luminance totale 7
3.6 Éclairage direct et ombres portées 8
3.7 Éclairage indirect 9
3.8 Textures 10
3.9 Correction Gamma 11

4 Calculs d'intersection **12**
4.1 Intersection rayon-sphère 12
4.2 Intersection rayon-plan 12
4.3 Intersection rayon-triangle 13
4.4 Intersection rayon-boîte englobante 13
4.5 Structure accélératrice 13

5 Conclusion **15**

1 Introduction

Dans le cadre de ce MOS, j'ai pu développer un algorithme de synthèse d'image par lancer de rayons (*Raytracing*). Le fonctionnement global de cette méthode s'articule autours des points suivants :

1. La scène à synthétiser est modélisée entièrement : objets, sources de lumière et caméra. Cela implique de réaliser de nombreux choix, comme par exemple le modèle de la caméra ou la description d'un objet.
2. Le comportement des objets vis-à-vis de la lumière est défini. Il est nécessaire de décrire, lorsqu'un rayon lumineux arrive en un point de la surface d'un objet, quelle quantité d'énergie va être ré-émise dans chacune des directions. Des modèles physiques plus ou moins complexes peuvent être employés.
3. Le calcul de la valeur de chacun des pixels de l'image est réalisé en lançant des rayons depuis la caméra, rayons qui rebondissent dans la scène et qui acquièrent ou perdent de l'énergie à chaque rebond.

Ce rapport a pour objectif de présenter l'ensemble des effets physiques que j'ai pu implémenter ainsi que la manière dont je les ai codés.

2 Scène

Une scène 3D intéressante comporte au minimum une caméra, une source de lumière et un objet dans le champ de vision de la caméra. Ces différents éléments ont été implémentés dans différentes classes. De plus, une classe `Vector` (pour les opérations vectorielles) et une classe `Ray` (pour les rayons lumineux) sont utilisées dans tout le code.

2.1 Objets

Plusieurs classes sont utilisées afin de modéliser différents types d'objets et de réaliser les calculs géométriques nécessaires :

- `Object` : une classe générale, utilisée pour exploiter le polymorphisme C++ qui permettra de considérer les objets `Sphere`, `Triangle` et `Geometry` comme du même type. Cette classe `Object` comporte des propriétés communes à tous les objets (couleur, albedo, transparence, miroir, intensité et indice optique). Les classes héritant de la classe `Object` sont les suivantes :
  - `Sphere` : une classe qui modélise une sphère, décrite par son centre et son rayon.
  - `Triangle` : une classe qui modélise un triangle, caractérisé par ses 3 sommets et sa normale.
  - `Geometry` : une classe qui modélise un maillage triangulaire, c'est-à-dire un assemblage de triangles. La méthode permettant de charger un maillage depuis un fichier `.obj` a été fournie. Une structure de données spatiale (de type *Bounding Volume Hierarchy*) est associée au maillage. La méthode de calcul de cette structure est décrite par la suite.
- `Scene` : une classe qui regroupe l'ensemble des objets de la scène et qui permet de déterminer l'illumination directe et indirecte, pour différents types de matériaux, en n'importe quel point de la scène.

2.2 Sources de lumière

Dans les premières versions du *Raytracer*, les sources de lumières étaient considérées comme ponctuelles (ce qui était physiquement incorrect). Puis elles ont ensuite été remplacées par des sphères dont le paramètre d'émissivité est strictement positif. Cela permet notamment de simuler des ombres douces, comme cela sera décrit par la suite.

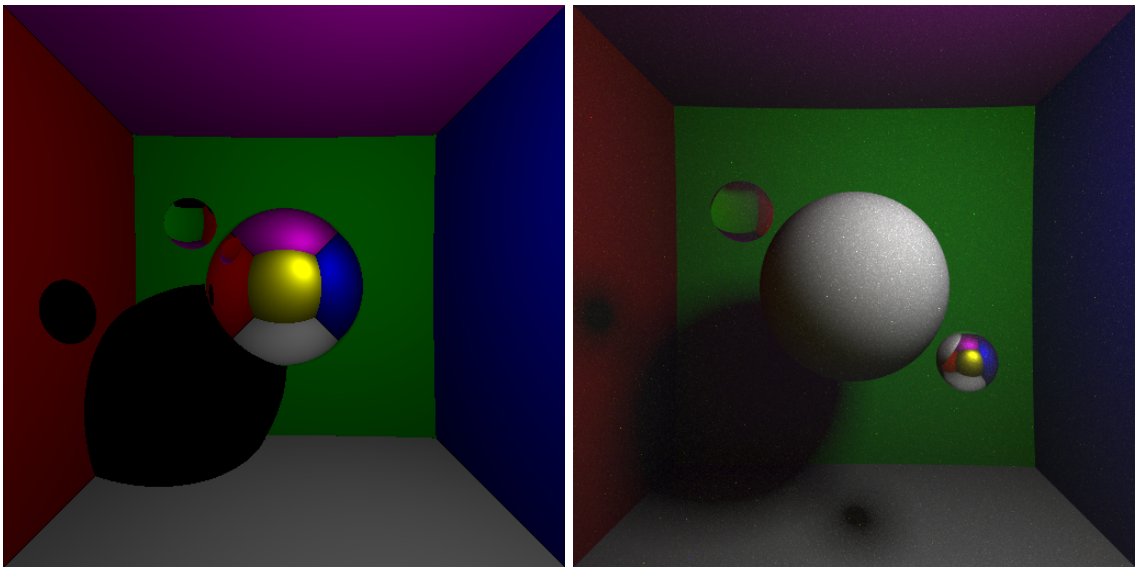

FIGURE 1 – (A gauche) Rendu avec source ponctuelle. (A droite) Rendu avec source sphérique.

2.3 Caméra

Le modèle de caméra utilisé est constitué d'un centre optique et d'une grille de pixels. Afin de déterminer la couleur de chacun des pixels, une certaine quantité N de rayons est lancée en partant du centre de la caméra et en passant par le pixel. Chaque rayon se déplace ensuite indépendemment des autres, rebondissant aléatoirement dans la scène afin d'estimer la quantité d'énergie perçue par le point réel représenté par le pixel. Deux précisions sont à ajouter concernant l'anti-aliasing et la profondeur de champ.

L'anti-aliasing consiste à éviter les contours crénelés. Pour cela, les rayons envoyés à travers un pixel partent tous du centre de la caméra mais ils passent ensuite par un point aléatoire du pixel (en utilisant une loi normale). Cela permet d'adoucir les bords.

La profondeur de champ, phénomène non intrinsèque au modèle de caméra utilisé, est simulé. Pour ce faire, une distance D à laquelle le plan net se situe ainsi qu'une ouverture o sont définies. Deux rayons sont ensuite lancés :

1. Un premier rayon artificiel partant du centre de la caméra.
2. Un second rayon partant proche du centre de la caméra (écarté d'une quantité aléatoire) et convergeant vers le même point que le premier rayon dans le plan net.

Ce léger décalage a pour conséquence de flouter les objets éloignés du plan net.

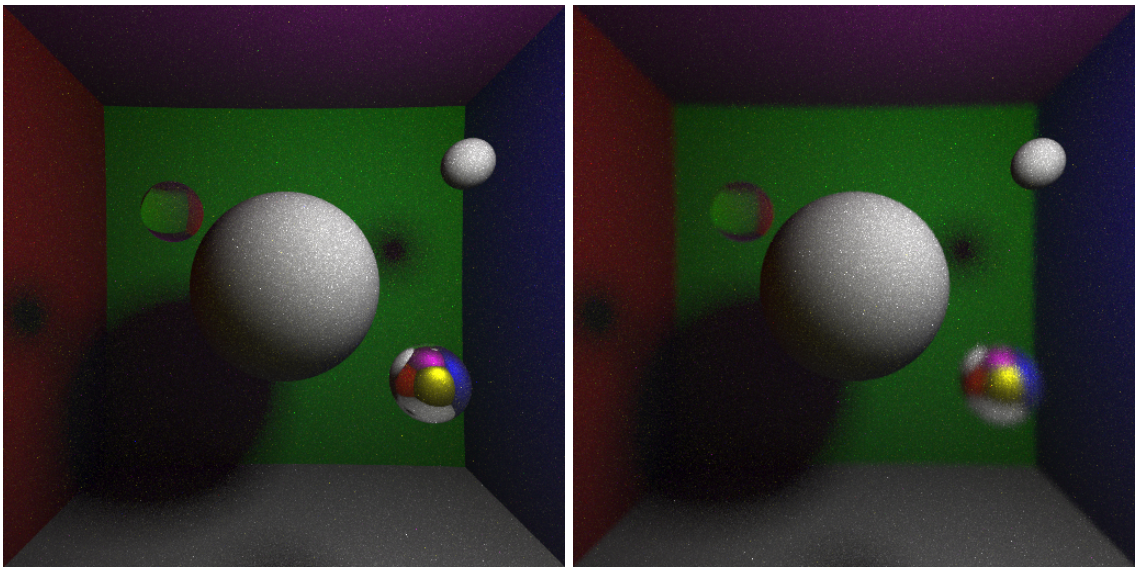

FIGURE 2 – (A gauche) Rendu sans effet de profondeur de champ. (A droite) Rendu avec effet de profondeur de champ.

3 Interaction lumière matière

Afin de produire un rendu réaliste, il est nécessaire de décrire précisément comment la lumière interagit avec la matière. Cela se fait par le biais d'une BRDF (Bidirectional reflectance distribution function) f définie en un point $\vec{p}$ comme le rapport de la luminance émise dans une direction $\vec{\omega}_{out}$ sur l'éclairement total en ce point $\vec{p}$:

$$f = \frac{L_{out}(\vec{p}, \vec{\omega}_{out})}{E(\vec{p})}$$

De plus, l'éclairement total en un point est obtenu par intégration de la luminance incidente en ce point :

$$E(\vec{p}) = \int_{\Omega} L_{in}(\vec{\omega}_{in}, \vec{p}) cos(\theta) d\vec{\omega}_{in}$$

Avec θ l'angle entre la normale au point $\vec{p}$ et le rayon incident $\vec{\omega}_{in}$. Il est donc possible d'exprimer la luminance émise par un point dans une direction donnée $\vec{\omega}_{out}$ uniquement à l'aide de la BRDF :

$$L_{out}(\vec{p}, \vec{\omega_{out}}) = \int_{\Omega} f(\vec{p}, \vec{\omega}_{in}, \vec{\omega}_{out}) L_{in}(\vec{\omega}_{in}, \vec{p}) cos(\theta) d\vec{\omega}_{in}$$

Enfin, par soucis de conservation de l'énergie, la BRDF doit être normalisée de manière à ce que l'énergie émise par un point soit inférieure ou égale à l'énergie perçue en ce point :

$$\int_{\Omega} f(\vec{p}, \vec{\omega}_{in}, \vec{\omega}_{out}) cos(\theta) d\vec{\omega}_{in} \leq 1$$

3.1 Matériau parfaitement diffus

Une manière simple de modéliser un certain type de matériau est de le considérer parfaitement diffus. Cela signifie que la luminance émise est isotrope : la même dans n'importe quelle direction. Sa BRDF est donc indépendante des directions considérées et la normalisation mène à :

$$f_{\text{diffus}}(\vec{p}, \vec{\omega}_{in}, \vec{\omega}_{out}) = \frac{cste}{\pi}$$

Cette BRDF a été implémentée dans la méthode `getColorLambert` de la classe `Scene`. Il s'agit du comportement par défaut de tous les objets.

3.2 Matériau parfaitement spéculaire

Une autre manière de modéliser certains matériaux est de les considérer comme parfaitement spéculaires, c'est-à-dire comme des miroirs parfaits. Ainsi, pour une direction $\vec{\omega}_{in}$ donnée, l'énergie est ré-émise entièrement dans la direction miroir $\vec{\omega}_{out}$. Cette direction miroir est obtenue par la relation suivante (les vecteurs sont considérés unitaires) :

$$\vec{\omega}_{out} - \vec{\omega}_{in} = 2\vec{n} \implies \vec{\omega}_{out} = \vec{\omega}_{in} + 2\vec{n}$$

La BRDF d'un matériau parfaitement spéculaire est donc la suivante :

$$f_{\text{spéculaire}}(\vec{p}, \vec{\omega}_{in}, \vec{\omega}_{out}) = \begin{cases} 1 \text{ si } \vec{\omega}_{out} = \vec{\omega}_{in} + 2\vec{n} \\ 0 \text{ sinon} \end{cases}$$

Cette BRDF a été implémentée dans la méthode `getColorMiroir` de la classe `Scene`. Un booléen `miroir` attribut de chaque objet permet de déterminer s'il possède ce comportement.

3.3 Matériau diffus et spéculaire : modèle de Phong

La quasi-totalité des matériaux présentent un comportement à la fois diffus et à la fois spéculaire. Le modèle de Phong modèlise ce mélange entre une contribution diffuse isotrope et une contribution spéculaire sous forme de lobe centré autour de la direction de réflexion parfaite. La BRDF du modèle est donc la suivante :

$$f_{\text{phong}}(\vec{p}, \vec{\omega}_{in}, \vec{\omega}_{out}) = \frac{\rho}{\pi} + (1-\rho)\frac{(n+2)}{2\pi} cos^n(\gamma)$$

Avec γ l'angle entre la direction de réflexion parfaite et la direction d'observation. Plus le paramètre n est élevé et plus le lobe sera étroit autour de la direction de réflexion parfaite. Le paramètre ρ détermine la part de chaque composante. Cette BRDF a été implémentée dans la méthode `getColorPhong` de la classe `Scene`. Un entier `n_phong` et un réel `rho_diff_phong` attributs de chaque objet permettent de déterminer s'il possède ce comportement.

3.4 Matériau transparent

Un matériau peut aussi être transparent. Dans ce cas sa BRDF est semblable à celle du matériau spéculaire (nulle partout sauf dans une direction) à la différence près que la direction $\vec{\omega}_{out}$ dépend cette fois aussi des indices optiques n_1 et n_2 des milieux traversés. La loi de Snell-Descartes donne :

$$n_1 sin(i_1) = n_2 sin(i_2)$$

Cette relation est ainsi utilisée pour déterminer, si il y a transmission, la direction $\vec{\omega}_{out}$ du rayon transmis. Si l'incidence est trop rasante, alors il y a réflexion et on utilise la méthode précédente de spécularité parfaite.

Cette BRDF a été implémentée dans la méthode `getColorTransparent` de la classe `Scene`. Un booléen `transparent` attribut de chaque objet permet de déterminer s'il possède ce comportement et, dans ce cas, l'indice optique de l'objet est précisé grâce à son attribut `indice_sphere`. L'indice optique de l'air dans la scène est pris égal à 1.0. Afin d'éviter un réflexion infinie à l'intérieur de l'objet, un seuil de récursion a été introduit.

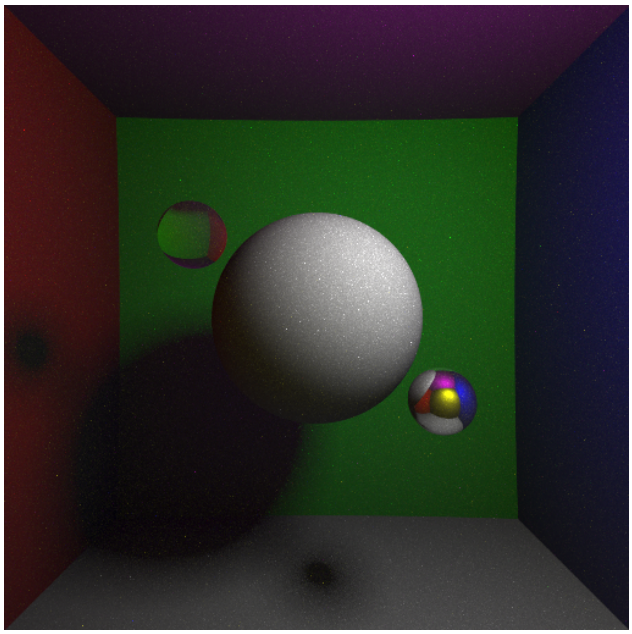

FIGURE 3 – De gauche à droite : une sphère transparente ($n = 1.33$), une sphère diffuse et une sphère parfaitement spéculaire

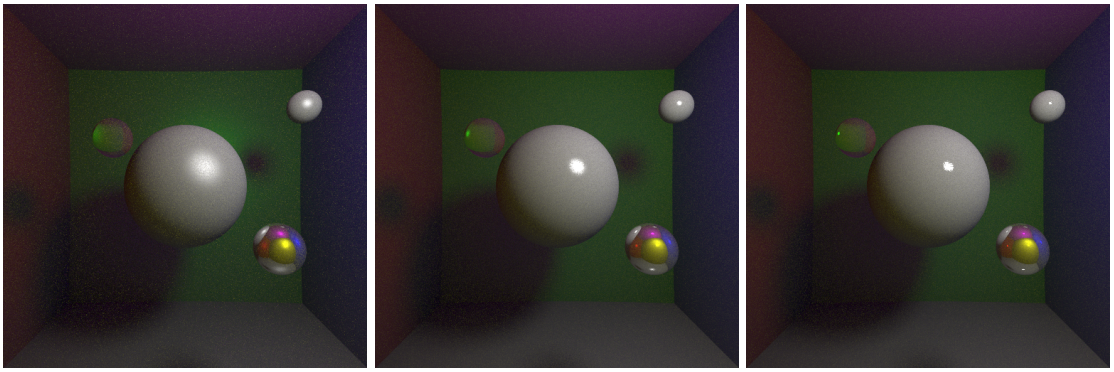

FIGURE 4 – Rendus avec BRDF de Phong. De gauche à droite : ($\rho = 0.7$,$n = 10$),($\rho = 0.7$,$n = 100$) et ($\rho = 0.7$,$n = 1000$)

3.5 Pixel et luminance totale

Physiquement, un rayon lumineux se propage en partant de la source, puis atteint un objet de la scène. Il est alors ré-émis dans plusieurs directions (suivant sa BRDF) et ainsi de suite, jusqu'à quitter la scène ou atteindre le capteur de la caméra. La part des rayons lumineux atteignant un des pixels du capteur de la caméra est infime. Il est donc plus efficace de réaliser le parcours du rayon lumineux en sens inverse : en lançant un rayon en partant de la caméra puis en le faisant rebondir dans la scène.

Les rayons lumineux que j'ai lancés depuis la caméra ont pour origine le centre C de la caméra (ou un point proche pour l'effet de profondeur de champ) et passant par P_{ij} le centre d'un pixel (ou un point proche pour l'anti-aliasing). Ce choix permet de générer un rendu en perspective. Si les rayons avaient eu pour origine le centre des pixels et tous la même direction (axe de la caméra), cela aurait permis de produire un rendu orthographique conservant les distances.

Lorsque le rayon lancé depuis la caméra intersecte un objet de la scène en un point $\vec{p}$, la couleur du pixel est déterminée par la luminance émise (pour chacune des couleurs primaires rouge, vert et bleu) dans la direction inverse du rayon, c'est à dire vers le pixel. Pour cela, il est nécessaire d'utiliser la relation liant luminance ré-émise, luminance perçue et BRDF présentée en introduction de cette partie, ainsi qu'un terme de source si nécessaire :

$$\begin{aligned}
L_{out}(\vec{p}, \vec{\omega_{out}} = p\vec{P}_{ij}) &= L_{\text{source}}(\vec{p}, \vec{\omega_{out}}) + \int_{\Omega} f(\vec{p}, \vec{\omega}_{in}, \vec{\omega}_{out}) L_{in}(\vec{\omega}_{in}, \vec{p}) cos(\theta) d\vec{\omega}_{in} \\
&= L_{\text{source}}(\vec{p}, \vec{\omega_{out}}) \\
&\quad + \int_{\Omega_{\text{source}}} f(\vec{p}, \vec{\omega}_{in}, \vec{\omega}_{out}) L_{in}(\vec{\omega}_{in}, \vec{p}) cos(\theta) d\vec{\omega}_{in} \\
&\quad + \int_{\Omega_{\text{non source}}} f(\vec{p}, \vec{\omega}_{in}, \vec{\omega}_{out}) L_{in}(\vec{\omega}_{in}, \vec{p}) cos(\theta) d\vec{\omega}_{in} \\
&= L_{\text{source}} + L_{\text{direct}} + L_{\text{indirect}}
\end{aligned}$$

Cette luminance perçue par le pixel en provenance du point $\vec{p}$ est composée de 3 termes principaux :

1. L_{source} : non nul uniquement si l'objet émet de la lumière, traduit par un attribut `intensite` strictement positif.
2. L_{direct} : correspond à la luminance reçue par le point $\vec{p}$ depuis les sources directes et ensuite ré-émise en direction du pixel, modulée par la BRDF.
3. L_{indirect} : correspond à la luminance reçue par le point $\vec{p}$ de manière indirecte (c'est à dire après rebond sur un objet de la scène) et ensuite ré-émise en direction du pixel, modulée par le BRDF.

3.6 Éclairage direct et ombres portées

Dans un premier temps, les sources avaient été considérées comme ponctuelles et le calcul de l'éclairage direct se faisait analytiquement. Lors du passage aux sources étendues, le calcul analytique n'est plus possible. Le calcul du terme L_{direct}, qui correspond à un calcul d'intégrale, se fait alors par estimation de Monte Carlo à la suite d'un changement de variable (obtenu par expression de l'angle solide en fonction de la surface source) :

$$\begin{aligned}
L_{\text{direct}}(\vec{p}, \vec{\omega_{out}}) &= \int_{\Omega_{\text{source}}} f(\vec{p}, \vec{\omega}_{in}, \vec{\omega}_{out}) L_{in}(\vec{\omega}_{in}, \vec{p}) cos(\theta) d\vec{\omega}_{in} \\
&= \int_{q \in S_{\text{source}}} f(\vec{p}, \vec{\omega}_{in} = \vec{qp}, \vec{\omega}_{out}) L_{in}(\vec{\omega}_{in} = \vec{qp}, \vec{p}) cos(\theta) cos(\alpha) \frac{dS}{||\vec{qp}||^2}
\end{aligned}$$

Avec α l'angle entre la normale à la source au point $\vec{q}$ et la direction $\vec{qp}$. On remarque que l'intensité lumineuse décroît en $\frac{1}{r^2}$, conformément à la propagation sphérique des ondes lumineuses. Il est important de préciser que, même si ce dernier n'apparaît pas dans les équations ci-dessus par soucis de clarté, un terme de visibilité a été introduit afin de déterminer si le point $\vec{p}$ est visible depuis le point $\vec{q}$.

Afin d'estimer la valeur de cette intégrale, j'ai utilisé la méthode de Monte Carlo, qui exprime une intégrale comme l'espérance d'une variable aléatoire $g(X)$ (exprimée

à partir d'une autre variable aléatoire X) et qui peut donc être approchée par une moyenne :

$$E(g(X)) = \int g(x) f_X(x) dx \simeq \frac{1}{N} \sum_{i=1}^{i=N} g(x_i) \implies g = \frac{f}{f_X} : \int f(x) dx \simeq \frac{1}{N} \sum_{i=1}^{i=N} \frac{f(x_i)}{f_X(x_i)}$$

Concrètement, cela revient à choisir aléatoirement des points sur la source de lumière, à calculer le terme sous l'intégrale associé à ce point modulé par la densité de probabilité d'avoir choisi ce point, puis à moyenner ces quantités. Cela a été mis en place pour une BRDF diffuse dans la méthode `getColorLambert` et pour une BRDF glossy dans la méthode `getColorPhong` de la classe `Scene`, l'opération de moyennage se faisant directement dans la fonction `main` du programme. Les autres BRDF n'ont pas besoin de cette estimation (cas de `getColorMiroir`) ou alors font appel aux autres BRDF (cas de `getColorTransparent`).

Le passage aux sources étendues permet d'obtenir des ombres douces (voir figure 1) puisque chaque point sera éclairé par une partie plus ou moins importante des sources. L'image devient beaucoup plus réaliste mais est bruitée à cause de l'estimation inexacte de l'intégrale par la méthode de Monte Carlo.

3.7 Éclairage indirect

Le calcul du terme L_{indirect}, lui aussi, correspond à un calcul d'intégrale. Il se fait alors par estimation de Monte Carlo, de manière analogue à l'estimation de L_{direct}. Toutefois, l'intégrale ne subit ici aucun changement de variable. Un rayon arrivant en $\vec{p}$ rebondit dans une direction aléatoire vers un point $\vec{q}$, et ainsi de suite. Après un nombre n_0 de rebonds fixé, Une estimation de l'éclairage direct du dernier point atteint est réalisée et cette quantité est ensuite modulée par les BRDF (et les autres termes sous l'intégrale) rencontrées lors du parcours. Cela est fait de manière récursive dans la méthode `getIndirect` de la classe `Scene`. La figure suivante présente un exemple de résultat obtenu.

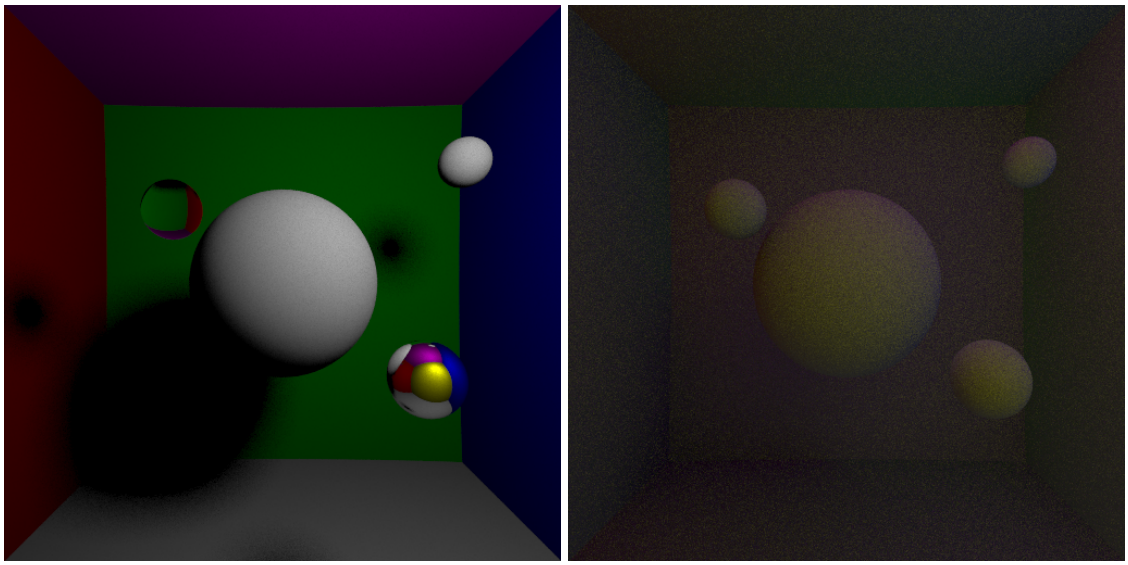

FIGURE 5 – (A gauche) : Eclairage direct uniquement. (A droite) : Eclairage indirect uniquement.

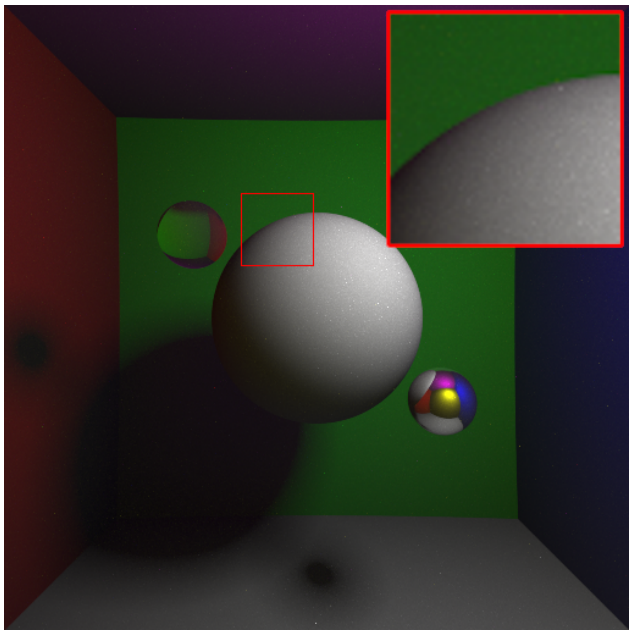

FIGURE 6 – Eclairage direct et indirect. La teinte violette de la sphère centrale est obtenue après rebond sur le plafond violet.

3.8 Textures

Les BRDF présentées précédemment permettaient de synthétiser des objets avec une couleur uniforme. Pour avoir plusieurs couleurs sur un même objet, il est possible de le discrétiser dans un maillage et d'associer une couleur à chacune des faces du maillage. Cependant, cette astuce atteint vite ses limites en ce qui concerne le niveau de détails atteignable. La solution à ce problème consiste à venir superposer une image (spécialement créée par un artiste), sur le maillage. Puis, lors de l'intersection entre un rayon et un maillage, il suffit de réaliser les étapes suivantes :

1. Déterminer la face intersectée.
2. Calculer les coordonnées barycentriques (par quotients d'aires) du point d'intersection au sein de la face.
3. En déduire les coordonnées de texture (u, v) du point d'intersection, par interpolation barycentrique à partir des coordonnées $(u, v)_i$ des sommets de la face.
4. Aller rechercher, dans l'image de texture, le pixel correspondant et sa couleur.

Le reste de l'algorithme est inchangé. Cette méthode a été mise en place dans la classe `Geometry`, notamment via la méthode `add_texture`. Voici un exemple de résultat :

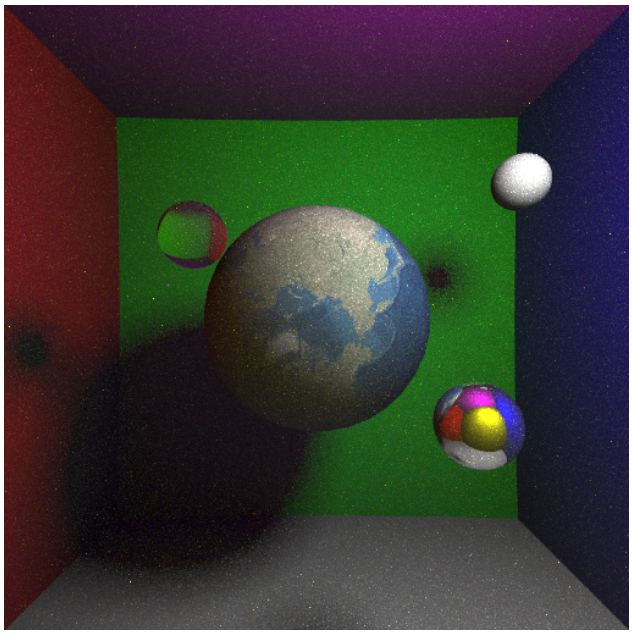

FIGURE 7 – Texture d'un planisphère plaquée sur un maillage de sphère.

3.9 Correction Gamma

Les images synthétisée dans le cadre de ce cours ont pour unique support un écran d'ordinateur. Cependant, ces écrans distordent les couleurs. Une correction gamma a donc été apportée. Cela consisté à modifier les valeurs des composantes (R,V,B) des pixels en les prenant à la puissance $\frac{1}{2.2} \simeq 0.45$. Ceci est réalisé juste avant l'enregistrement de l'image, à la fin du `main`.

4 Calculs d'intersection

Un élément important en synthèse d'image par lancer de rayons est le calcul de l'intersection entre un rayon (caractérisé par son origine et sa direction) et la scène (un ensemble de géométries positionnées dans l'espace). A chaque géométrie correspond une manière différente de déterminer si, oui ou non, un rayon donné intersecte la géométrie et, en cas d'intersection, déterminer le point d'intersection. Pour implémenter cela dans l'algorithme, j'ai utilisé :

- Une méthode virtuelle `intersect` de la classe `Object`. Chaque classe héritant de la classe `Object` (c'est le cas pour les classes `Sphere`, `Triangle` et `Geometry`) réécrit cette méthode en accord avec la géométrie qu'elle représente. Ainsi, il est ensuite possible de considérer n'importe quel objet (parmi `Sphere`, `Triangle` et `Geometry`) et la méthode adaptée sera appelée.
- Une structure C++ `intersection_details` qui contient différents types d'informations à propos d'une intersection :
  - Des informations géométriques : le point d'intersection (`inter_Pos`), la normale au point d'intersection (`inter_Norm`), etc.
- Des informations concernant le matériau : la couleur (`inter_Color`), l'albedo (`albedo`), la BRDF (`miroir`, `transparent`,`indice_sphere`,`n`$_p hong$), *etc.* Ces informations sont ensuite utilisées dans les méthodes de calcul d'éclairage direct et indirect présentées précédemment.

4.1 Intersection rayon-sphère

Le calcul de l'intersection entre un rayon (origine O et direction $\vec{u}$) et une sphère (centre C rayon R) se fait par résolution d'un polynôme du second ordre :

$$\begin{aligned} & ||O + \lambda\vec{u} - C||^2 = R^2 \\ \iff \quad & ||\vec{CO}||^2 + ||\lambda\vec{u}||^2 + 2(\vec{CO}|\lambda\vec{u}) = R^2 \\ \iff \quad & \lambda^2||\vec{u}||^2 + 2\lambda(\vec{CO}|\vec{u}) + ||\vec{CO}||^2 - R^2 = 0 \end{aligned}$$

Le calcul du discriminant permet de connaître le nombre de points d'intersection (0,1 ou 2) ainsi que leurs positions. La normale à un point P à la surface de la sphère est donnée par le vecteur $\vec{CP}$ normalisé. La méthode `intersect` de la classe `Sphere` met en place ce calcul.

4.2 Intersection rayon-plan

Le calcul de l'intersection entre un rayon (origine O et direction $\vec{u}$) et un plan (point P et normale $\vec{n}$) est nécessaire pour le calcul de l'intersection entre un rayon et un triangle. Cela se traduit par :

$$\begin{aligned} O + \lambda\vec{u} \in (P, \vec{n}) \quad & \iff \quad (O + \lambda\vec{u} - P|\vec{n}) = 0 \\ & \iff \quad (\vec{PO}|\vec{n}) + \lambda(\vec{u}|\vec{n}) = 0 \\ & \iff \quad (\vec{u}|\vec{n}) \neq 0 \text{ et } \lambda = \frac{(\vec{OP}|\vec{n})}{(\vec{u}|\vec{n})} \\ & \qquad\quad \text{ou } (\vec{u}|\vec{n}) = 0 \text{ et } (\vec{OP}|\vec{n}) = 0 : \text{rayon dans plan} \end{aligned}$$

La valeur du paramètre λ permet d'obtenir le point d'intersection précis. La normale au point d'intersection est connue puisqu'il s'agit de la normale au plan.

4.3 Intersection rayon-triangle

Le calcul de l'intersection entre un rayon (origine O et direction $\vec{u}$) et un triangle (trois points orientés A,B et C et normale $\vec{n} = \vec{AB} \wedge \vec{AC}$) s'appuie sur le calcul de l'intersection entre le rayon et le plan $(A, \vec{n})$. Une fois le point d'intersection avec le plan obtenu, il suffit de déterminer si ce point d'intersection P est à l'intérieur du triangle. Cela se fait par l'utilisation des coordonnées barycentriques :

$$\begin{aligned} & \vec{AP} = \beta\vec{AB} + \gamma\vec{AC} \\ \Longrightarrow \quad & (\vec{AP}|\vec{AB}) = \beta||\vec{AB}||^2 + \gamma(\vec{AC}|\vec{AB}) \\ & (\vec{AP}|\vec{AC}) = \beta(\vec{AB}|\vec{AC}) + \gamma||\vec{AC}||^2 \end{aligned}$$

La résolution de ce système linéaire de 2 équations à 2 inconnues se fait par la méthode de Cramer. Puis il suffit ensuite de vérifier que $0 \leq \beta \leq 1$,$0 \leq \gamma \leq 1$ et $0 \leq \beta + \gamma \leq 1$. La méthode `intersect` de la classe `Triangle` met en place ce calcul.

4.4 Intersection rayon-boîte englobante

L'intersection entre un rayon (origine O et direction $\vec{u}$) et une boîte englobante alignée avec les axes $(x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max})$ est utilisée dans les structures accélératrices. Pour cela, on utilise la méthode d'intersection entre un rayon et chacun des 6 plans de la boîte. Cela permet d'obtenir 3 paires de valeurs pour le paramètre λ :

1. $\lambda_{x,1}$ et $\lambda_{x,2}$ pour les plans de normale $\vec{e}_x$, réordonnées en $\lambda_{x,min} = min(\lambda_{x,1}, \lambda_{x,2})$ et $\lambda_{x,max} = max(\lambda_{x,1}, \lambda_{x,2})$.
2. $\lambda_{y,1}$ et $\lambda_{y,2}$ pour les plans de normale $\vec{e}_y$, réordonnées en $\lambda_{y,min} = min(\lambda_{y,1}, \lambda_{y,2})$ et $\lambda_{y,max} = max(\lambda_{y,1}, \lambda_{y,2})$.
3. $\lambda_{z,1}$ et $\lambda_{z,2}$ pour les plans de normale $\vec{e}_z$, réordonnées en $\lambda_{z,min} = min(\lambda_{z,1}, \lambda_{z,2})$ et $\lambda_{z,max} = max(\lambda_{z,1}, \lambda_{z,2})$.

Ainsi, il y a intersection si et seulement si les intervalles $[\lambda_{-,min}, \lambda_{-,max}]$ se chevauchent, ce qui équivaut à :

$$max(\lambda_{x,min}, \lambda_{y,min}, \lambda_{z,min}) \geq min(\lambda_{x,max}, \lambda_{y,max}, \lambda_{z,max}) \iff \text{intersection}$$

Ici, la position du point d'intersection ne nous intéresse pas. La méthode `intersection` de la classe `BBox` met en place ce calcul.

4.5 Structure accélératrice

Un maillage est décrit comme un assemblage de triangles, souvent plusieurs centaines et même milliers. Le test d'intersection entre un rayon et chacun de ces triangles peut devenir très long. C'est pourquoi une structure accélératrice de type *Bounding Volume Hierararchy* (BVH) a été mise en place. Un BVH consiste ici en une description spatiale d'un maillage sous forme d'arbre de boîtes englobantes. En

partant de la boîte englobant le maillage complet (racine de l'arbre), on subdivise récursivement chaque noeud de l'arbre en deux boites englobantes suivant la direction la plus grande. Deux boîte soeurs dans l'arbre peuvent se chevaucher mais elles occupent globalement chacune la moitié de l'espace de leur parent. Ce processus s'arrête lorsqu'une boîte contient moins de 10 triangles (choix arbitraire).

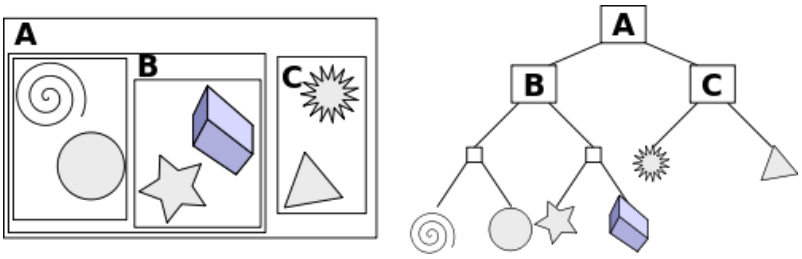

FIGURE 8 – Schéma de principe du BVH (source : `https://en.wikipedia.org/wiki/Bounding_volume_hierarchy`)

L'arbre est stocké dans une structure C++ `node` contenant les indices de début et de fin de ses triangles triés, ainsi que des pointeurs vers ses fils (eux aussi des `node`).

La méthode `build_hierarchy` de la classe `Geometry` permet de réaliser cette construction récursive du BVH. Ainsi, lors du calcul de l'intersection entre un rayon et un maillage (méthode `intersect` de la classe `Geometry`), il n'est plus nécessaire de tester chacun des triangles du maillage. Il suffit de parcourir l'arbre en largeur si le parent a été intersecté, puis de tester l'intersection des triangles des feuilles terminales (soit une dizaine seulement). Les gains en performances sont importants grâce à cette structure, ce qui a permis d'obtenir l'image suivante assez rapidement :

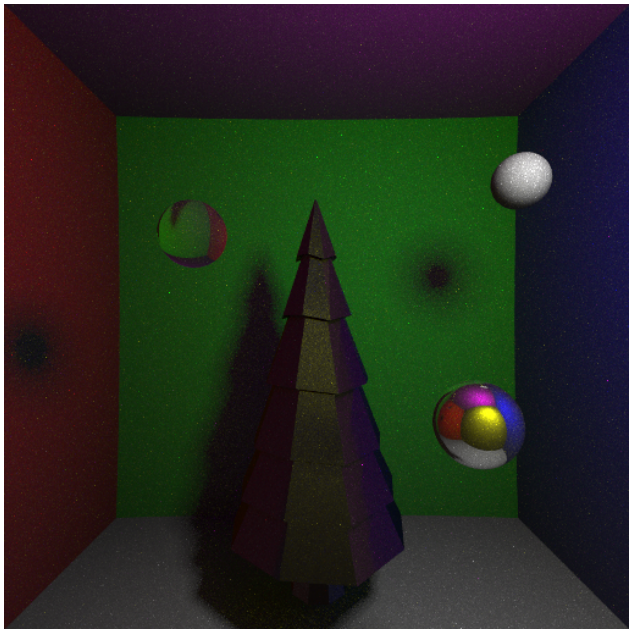

FIGURE 9 – Le sapin est décrit par un maillage et intersecté grâce à une structure accélératrice.

5 Conclusion

Ce cours m'a permis de développer un algorithme de synthèse d'images par lancer de rayon simple. Il permet de générer le rendu d'une grande diversité d'objets (grâce notamment au support de maillages) et de conditions d'éclairement (via la disposition libre de sources de lumière sphériques). Seuls quelques comportements de matériaux (parfaitement diffus, parfaitement spéculaire, diffus/spéculaire et transparent) et quelques effets optiques (anti-alising, profondeur de champ) ont été développés mais cette structure peut servir de base pour l'ajout d'autres effets physiques : subsurface scattering, BRDF anisotrope, rendu de fumée, fourrure, etc.