Rapport projet 2 :
Lancer de rayon

Clara BÉGUÉ, Maxime CAUTÉ, Odile RADET

ENS Rennes

7 décembre 2018

Résumé

Dans ce projet, nous avons implémenté un algorithme de lancer de rayon. Pour cela, nous avons tout d'abord mis en place une structure de classe très modulaire pour pouvoir facilement implémenter des objets. Puis, nous avons mis en place le modèle de Phong pour modéliser les effets de lumières.

Table des matières

Introduction

L'objectif de ce projet est de modéliser les effets d'une source lumineuse sur une scène. Nous nous sommes appuyés sur le principe du retour inverse de la lumière ainsi que sur le modèle de Phong [1]. A la fin du projet, nous avons réussi à obtenir des images telles que celle présentée à la figure 1.

1 Principe de base

Les algorithmes implémentés se basent principalement sur l'optique géométrique, la lumière est supposée être constituée de rayons rectilignes, et sur le principe de retour inverse de la lumière.

Le principe est d'envoyer des rayons depuis une caméra, d'identifier la couleur des objets rencontrés sur son chemin en utilisant principalement le modèle de Phong et d'afficher la couleur obtenue à l'écran, comme sur la figure 2.

2 Organisation du code

2.1 `Source` et `Lightray`

Nous avons implémenté des sources et des rayons afin de modéliser la lumière.

Les sources, dans leur implémentation, ne sont que des sources ponctuelles, nous n'avons pas considéré les sources étendues. Elles sont définies par leur position ainsi que leurs couleurs.

Pour faire la liaison entre les sources et l'écran, nous avons implémenté des rayons lumineux, dans une classe `Lightray`. Un rayon est défini par son origine et sa direction.

Il contient en outre des informations de couleurs, et d'autres paramètres lumineux.

2.2 Structure virtuelle `Shape`

L'implémentation des objets est réalisée au travers d'une classe abstraite `Shape`. Le diagramme des classes liés à `Shape` est présenté figure 3. Cette structure contient toutes les informations partagées par les objets de la scène :

- leur position (un point permettant de définir l'ensemble de ceux de l'objet) ;
- leurs caractéristiques physiques (dans notre implémentation seulement celles liées aux effets lumineux).

Il est à noter que les caractéristiques physiques auraient pu être regroupées dans une classe `Material`.

En outre, cette classe contient l'ensemble des objets de la scène dans la variable `static vector<Shape*>`.

Deux méthodes virtuelles sont héritées et implémentées par ses héritiers :

- `check_intersection` qui prend en entrée un rayon et renvoie l'impact de celui-ci sur la forme (ou `No_hit` si il n'y a pas d'impact) ;
- `reflected_ray` qui prend en entrée un rayon et renvoie le rayon réfléchi par la forme (ou lui-même si il n'y a pas de réflexion).

L'implémentation de ces méthodes dans les classes filles sera traitée séparément.

1. http ://rodolphe-vaillant.fr/images/courses/phong_shading.pdf

2.3 Boules

Les boules sont implémentées en héritage de la classe `Shape`, par une classe `Ball`.

Elles sont repérées par leur centre (attribut `position` hérité de la classe Shape), ainsi que leur rayon.

2.4 Plans

2.4.1 Structure virtuelle `Plane`

Les plans sont héritiers d'une classe abstraite `Plane` qui représente les plans dans leur caractère mathématique.

Ils sont repérés par un point qu'ils contiennent (`position`) et un vecteur normal à eux.

Nous avons implémenté une méthode supplémentaire `check_intersection` qui renvoie le point d'impact d'un rayon passé en paramètre avec le plan.

Un tel point existe toujours de manière unique sur un plan mathématique (infini), à moins que le rayon ne soit parallèle au plan, auquel cas nous renvoyons `No_point`.

Le calcul de ce point se fait de la manière suivante (nous noterons O l'origine du rayon, $\overrightarrow{d}$ sa direction (vecteur unitaire), P un point donné du plan, $\overrightarrow{n}$ un vecteur normal au plan :

1. on vérifie si le rayon est parallèle au plan, i.e. si $\overrightarrow{d} \cdot \overrightarrow{n} = 0$, auquel cas on renvoie `No_point;`
2. on évalue la distance de l'origine du rayon au point d'intersection, qui vaut $l = \frac{\overrightarrow{OP} \cdot \overrightarrow{n}}{\overrightarrow{d} \cdot \overrightarrow{n}}$;
3. on vérifie que l'impact se fait bien dans le sens suivi par le rayon, i.e. que l est positif, sinon on renvoie `No_hit;`
4. on renvoie $O + l \times \overrightarrow{d}$ qui est bien le point d'impact.

Pour la correction du calcul de l, en notant I le point d'intersection, nous avons $l \times \overrightarrow{d} = \overrightarrow{OI}$ ($\overrightarrow{d}$ est unitaire). Or I et P sont dans un plan normal à $\overrightarrow{n}$, d'où $\overrightarrow{OI} \cdot \overrightarrow{n} = \overrightarrow{OP} \cdot \overrightarrow{n}$.

2.4.2 Plans infinis

Les plans infinis pourraient être des instances de la classe `Plane`. Toutefois, nous avons pris le parti de créer une classe fille `Infinite_Plane` qui implémente simplement les méthode virtuelle héritées indirectement de `Shape`. `Plane` reste ainsi une classe virtuelle, non implémentable.

L'intérêt de cette manœuvre est double : nous conservons tout d'abord une structure d'héritage où seules les classes abstraites sont héritées, en modularisant ainsi le code, mais surtout nous conservons une structure théorique (les plans génériques, mathématiques) distincte d'une structure physique (le plan infini). Cette division nous permet ainsi de relier tous les objets plans au travers de cette classes abstraite.

Si nous aurions ainsi pu multiplier les objets en 2 dimension grâce à cette hiérarchie, nous nous sommes contentés d'implémenter les plans finis.

2.4.3 Plans finis

Les plans finis (des parallélogrammes, géométriquement parlant) héritent également de `Plane`.

Toutefois, ils nécessitent une définition plus précise : il faut connaître un point, deux vecteurs directeurs unitaires pour la hauteur et la largeur, et deux valeurs pour celles-ci (ou des données équivalentes).

Nous avons fait ce choix, qui ne nécessite pas le vecteur normal, afin de simplifier la manipulation de ces objets. Pour les mêmes raisons, nous avons décidé de contenir les information de hauteur et de largeur dans les deux vecteurs directeurs, en leur ôtant donc le caractère unitaire.

Ainsi, l'utilisateur n'a besoin d'entrer qu'un point du plan fini, et deux vecteurs permettant de tracer les deux côtés du plan touchant ce point, pour créer le plan. Cette solution permet donc de manipuler des données facilement visualisables. Le calcul du vecteur normal se fait à partir de ces informations (produit vectoriel normalisé des deux vecteurs).

3 Détection des formes

3.1 Calcul du rayon réfléchi

Lorsqu'un rayon intersecte une forme présente (ce que l'on détermine grâce aux méthodes détaillées dans la partie suivante), on détermine le rayon réfléchi par cette forme : il s'agit du rayon dont l'origine est le point d'intersection, et dont la direction est le vecteur symétrique du vecteur direction du rayon incident par rapport à un vecteur normal à la forme au point d'intersection.

En notant $\overrightarrow{i}$ le vecteur directeur du rayon incident, $\overrightarrow{r}$ le vecteur directeur du rayon réfléchi, I le point d'intersection et $\overrightarrow{n}$ le vecteur normal (unitaire) en I à la forme sur laquelle le rayon incident est réfléchi, on a :

$$\overrightarrow{r} = \overrightarrow{i} - 2 \cdot \overrightarrow{n}.$$

3.2 Calcul du point d'impact

On va expliquer dans cette partie, dans le cas des trois types de formes implémentés, comment nous avons effectué le calcul du point d'impact (lorsque celui-ci existe).

Cas d'une sphère Considérons ici une sphère de centre $C(x_C, y_C)$ et de rayon R. On note A le point d'origine du rayon incident, et $\overrightarrow{u}$ le vecteur directeur de ce rayon.

L'équation de la sphère étant $M(x,y) \in S(C,R) \Leftrightarrow (x - x_C)^2 + (y - y_C)^2 = R^2$, on commence par résoudre l'équation suivante :

$$\left|\left|\overrightarrow{AC} - t \times \overrightarrow{u}\right|\right| = R.$$

On choisit ensuite la plus petite solution positive t_0. Le point d'intersection est alors le point à l'extrémité du vecteur $\overrightarrow{AC} - t_0 \times \overrightarrow{u}$.

Cas d'un plan infini On commence par tester l'existence d'un point d'intersection en calculant le produit scalaire d'un vecteur normal au plan et d'un vecteur directeur du rayon incident. Si celui-ci est nul (à ε près, où ε représente la précision voulue), alors la droite est parallèle au plan et ne l'intersecte donc pas. Dans le cas contraire, le point d'intersection existe.

On calcule ensuite l'intersection comme suit.

On pose $d = -\dfrac{\overrightarrow{u} \cdot \overrightarrow{n}}{\overrightarrow{n} \cdot \rightarrow}$ où $\overrightarrow{u}$ est le vecteur $\overrightarrow{PO}$ avec P la position du plan et O l'origine du rayon, et avec $\overrightarrow{n}$ vecteur normal au plan.

Enfin, le point d'intersection I est le point $O + d \times \overrightarrow{u}$.

Il faut cependant vérifier que le point d'intersection est situé du même côté du plan que l'origine du rayon. Pour cela, tant que ce n'est pas le cas, on diminue la distance du point d'intersection au point d'origine du rayon.

Pour savoir si les deux points sont du même côte du plan, il faut tester si, avec P point `position` du plan, I point d'intersection et O point d'origine du rayon, le projeté des vecteurs $\overrightarrow{PI}$ et $\overrightarrow{PO}$ sur la normale au plan sont de même signe. Dans le code rendu, il y avait une erreur à cet endroit là. Dans plane2.cc, à la ligne 53, il fallait écrire :

```
correct_side = sign_intersection_normal*sign_ray_normal > 0;
```

au lieu de :

```
correct_side = sign_intersection_normal/sign_ray_normal > 0;
```

Cas d'un plan fini Dans le cas d'un plan fini, on commence par déterminer l'intersection du rayon incident et du plan infini contenant le plan fini considéré, comme dans le paragraphe précédent. Il reste seulement à vérifier que ce point d'impact potentiel est bien dans les limites du plan fini.

Pour cela, il suffit de calculer le produit scalaire de $\overrightarrow{PI}$ (où P est la position, donc le centre du plan fini, et I le point d'intersection potentiel) avec chacun des deux vecteurs qui donnent les deux directions de plan, puis de comparer les valeurs absolues de ces deux produits scalaires avec la demi-longueur et la demi-largeur du plan, afin de vérifier que I est bien dans les limites fixées du rectangle.

4 Effets de lumière

4.1 Synthèses de la lumière

La première chose que nous avons dû faire pour faire apparaître les jeux de lumière fut de décider comment calculer l'absorption par les objets ainsi que la synthèse additive de deux rayons lumineux.

4.1.1 Absorption par les objets

Pour l'absorption, pour chaque composante rouge, verte et bleue,nous avons choisi de prendre le minimum entre la composante de la lumière incidente et la composante de la couleur de l'objet. En notant la couleur de la lumière $(\text{red}_l, \text{green}_l, \text{blue}_l)$, celle de l'objet $(\text{red}_o, \text{green}_o, \text{blue}_o)$ et la couleur résultante $(\text{red}, \text{green}, \text{blue})$, on obtient :

$$\begin{array}{lcl} \text{red} & = & \min(\text{red}_l,\ \text{red}_o) \\ \text{green} & = & \min(\text{green}_l,\ \text{green}_o) \\ \text{blue} & = & \min(\text{blue}_l,\ \text{blue}_o) \end{array}$$

Par la suite, cette opération sera notée $+_{\text{ab}}$.

4.1.2 Synthèse additive

Afin de modéliser la synthèse additive de la lumière, nous avons choisi d'additionner les composantes rouges, vertes et bleues, en bornant la somme à 255. Par la suite, cette addition sera notée $+_a$.

4.2 Modèle de Phong

Pour calculer les ombres et les effets de diffusion de la lumière, nous avons utilisé le modèle de Phong. Ce dernier est un modèle local, qui s'applique pixel par pixel sans prendre en considération les couleurs voisines, et décompose l'action de la lumière en trois composantes : lumière ambiante, diffuse et spéculaire.

4.2.1 Lumière ambiante

La première composante est la lumière ambiante. Elle correspond à la luminosité générale de la scène.

Pour la représenter, nous avons implémenté :

— une lumière ambiante, qui correspond à la couleur du fond `ambient_color` ;

— un coefficient d'absorption $k_a \in [0, 1]$ propre au milieu (pour un rendu plus réaliste, ce coefficient est souvent fixé entre 0.1 et 0.2).

La luminosité ambiante correspond à `ambient_color*ka`. Plus `ambient_color` et k_a sont élevés et plus la scène et les objets vont paraître lumineux.

On calcule la couleur des objets associés à la luminosité ambiante en calculant l'absorption de cette dernière par les objets. Vous pouvez voir un exemple à la figure 5.

4.2.2 Lumière diffuse

La deuxième composante est la lumière diffuse. Elle correspond au dégradé de la lumière issus des sources à la surface des objets.

Pour la représenter, nous avons ajouté un coefficient de diffusion $k_d \in [0, 1]$ propre à chaque objet. L'intensité de la lumière en un point de l'objet est proportionnelle à cosinus de l'angle que fait le rayon lumineux issus de la source avec la normale à la surface de l'objet, soit l'angle θ sur la figure 4.

Ainsi, en notant la couleur de l'objet couleur_o et couleur_s celle d'une source, pour obtenir la couleur de l'objet due à la diffusion, on fait :

$$\text{diffusion} = \text{couleur}_o +_{\text{ab}} k_d \times \sum_{s \text{ source}} \cos\theta_s \times \text{couleur}_s$$

Plus k_d est élevé et plus le dégradé de la lumière sera marqué. Les effets de la diffusion seule sont montrés à la figure 6.

4.2.3 Lumière spéculaire

Le dernier effet du modèle de Phong est la lumière spéculaire, c'est-à-dire le reflet de la source lumineuse sur les objets.

Pour la représenter, nous avons introduit deux paramètres propres à chaque objet :

— le coefficient de spécularité $k_s \in [0, 1]$, qui traduit l'intensité de la tâche (elle augmente avec k_s) ;

— le coefficient de Phong $\alpha \geq 1$, qui traduit le diamètre de la tâche (il diminue quand α augmente).

La composante spéculaire de la lumière est proportionnelle à $\cos\Omega^\alpha \times k_s$, où Ω est défini sur la figure 4. La composante spéculaire faut donc :

$$\text{spéculaire} = k_s \times \sum_{s \text{ source}} \cos\Omega^\alpha \times \text{couleur}_s$$

Les effets de la lumière spéculaire seule sont visibles à la figure 7.

Finalement, en superposant ces trois effets, on obtient la figure 8. En notant couleur_a la couleur ambiante, la couleur de l'objet perçue par l'observateur sera :

$$\text{couleur} = \text{spéculaire} +_a \ [\text{couleur}_o +_{\text{ab}} (k_d \times \sum_{s \text{ source}} \cos\theta_s \times \text{couleur}_s \ +_a \ \text{couleur}_a \times k_a)]$$

4.3 Baisse d'intensité sur les zones de lumière spéculaire de la source

Nous avons remarqué lors de nos tests un phénomène étrange : au lieu de la tâche spéculaire, apparaissait souvent une zone plus pâle entouré d'un anneau lumineux plus clair comme à la figure 9(a). Une analyse des couleurs montre bien que la zone centrale est plus pâle (de couleur environ (225,0,0) au centre et (240,0,0) en périphérie). Ce phénomène apparaît lors de la superposition des effets de diffusion et de spécularité. Après quelques tests, il nous est apparu que c'était dû au modèle de synthèse additive utilisé, bien trop simpliste. En le remplaçant par :

$$\text{couleur} = \sqrt{\frac{\text{couleur}_1^2}{2} + \frac{\text{couleur}_2^2}{2}}$$

le problème est résolu, comme on peut le constater sur la figure 9(b). Cependant, avec ce modèle, les couleurs obtenues étaient bien plus sombres à cause du facteur $\frac{1}{2}$, ce qui nous obligeait à fortement augmenter le coefficient ambiant, au-delà de 1. La mise en place d'un modèle plus adapté et réaliste est donc une piste fortement envisageable pour une potentielle extension du projet.

4.4 Réflexion

Enfin, nous avons essayé d'implémenter de la réflexion. Le principe est simple : il s'agit de déterminer la couleur du rayon réfléchi de manière récursive et d'ajouter cette couleur fois k_s à celle obtenue précédemment grâce à une synthèse additive de la lumière. Le nombre de réflexion limite a été fixé à deux, car au-delà les effets de la réflexion n'était presque plus visibles.

Ainsi, nous avons pu créer des miroirs en créant des formes noires telles que $k_d = 0$, $k_s = 1$ et $\alpha = 100$.

Nous avons réussi à implémenter la réflexion après le rendu des codes mais la figure 10 est un exemple de ce que avons pu obtenir pour des surfaces réflexives et la figure 11 montre des miroirs.

Conclusion

En conclusion, nous sommes parvenus à implémenter une structure de forme très modulaire qui peut aisément être étendu, avec des triangles par exemple, pour pouvoir représenter des formes bien plus complexes, ainsi que le modèle de Phong et un début de réflexion. En complément, nous aurions pu créer une classe `Material` afin de reproduire des textures de manière opaque. De plus, il était nécessaire de mettre en place un synthèse additive de la lumière plus réaliste. Enfin, nous aurions pu fusionner les classes `Source` et `Shape` ou étendre la classe `Source`, afin d'obtenir des sources étendues, bien plus réalistes.

Annexes

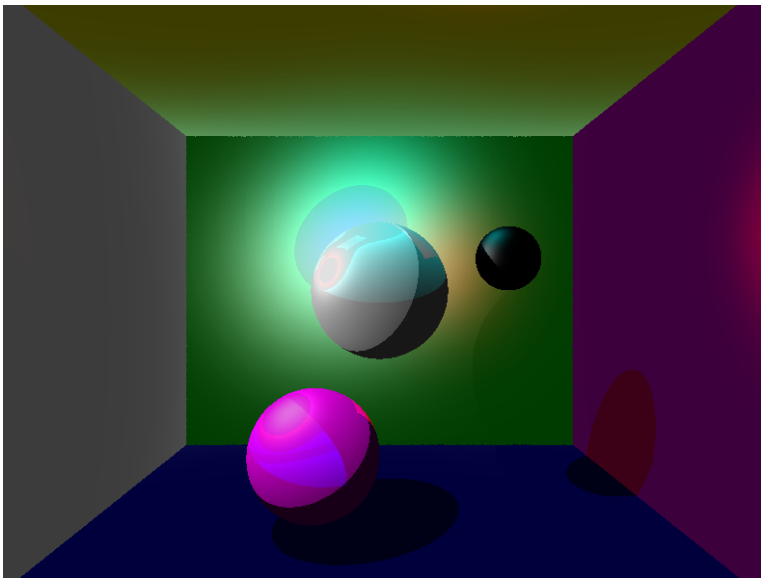


FIGURE 1 – Exemple de résultat avec 3 boules entre 6 plans éclairés par 3 sources de lumières

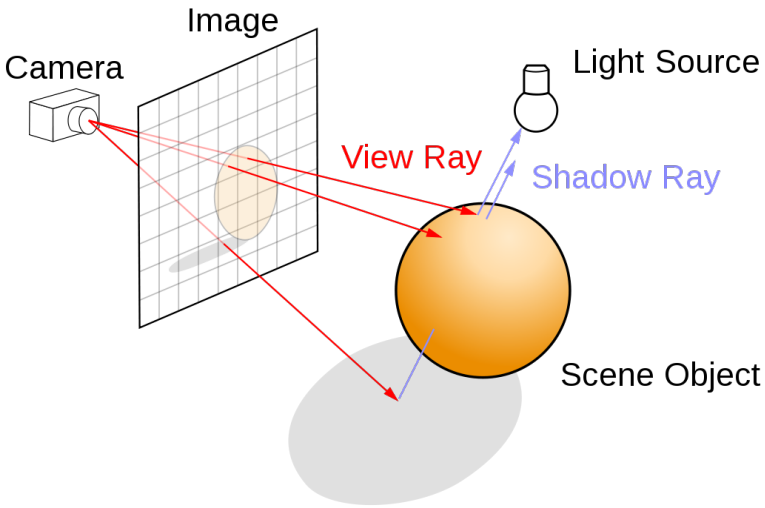


FIGURE 2 – Principe du lancer de rayon [2]

2. https ://fr.wikipedia.org/wiki/Ray_tracing

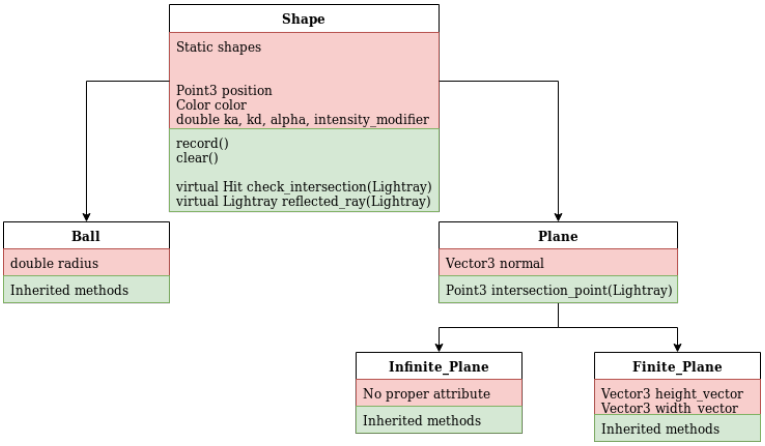


FIGURE 3 – Diagramme des classes

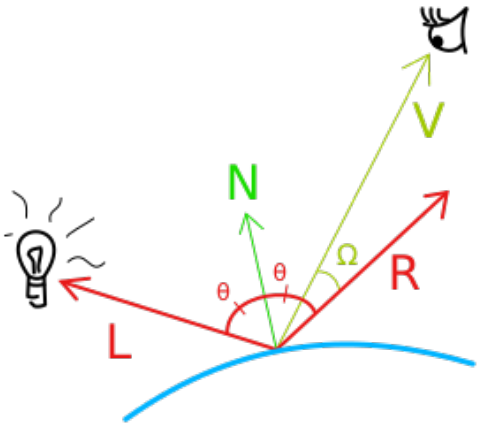


FIGURE 4 – Incidence de la lumière sur un objet

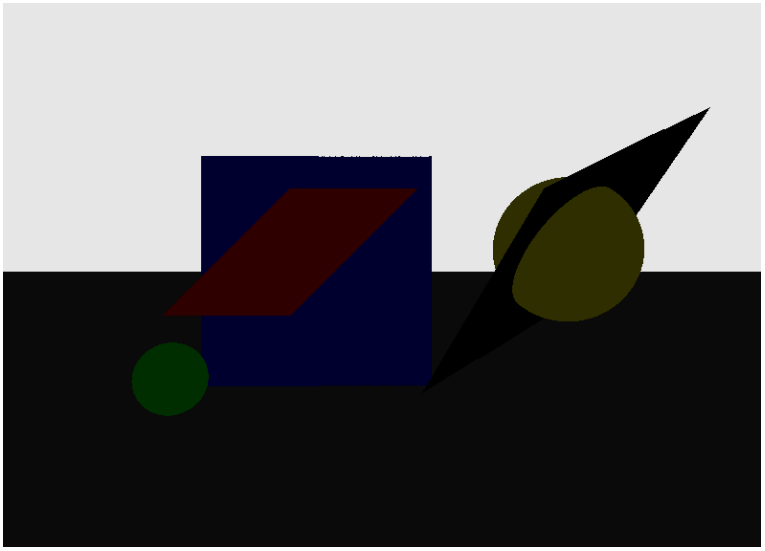


FIGURE 5 – Luminosité ambiante seule

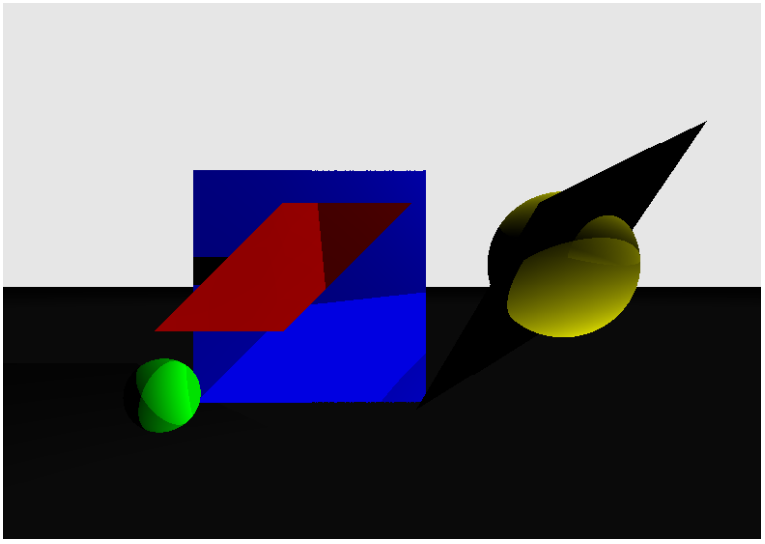


FIGURE 6 – Lumière diffuse seule

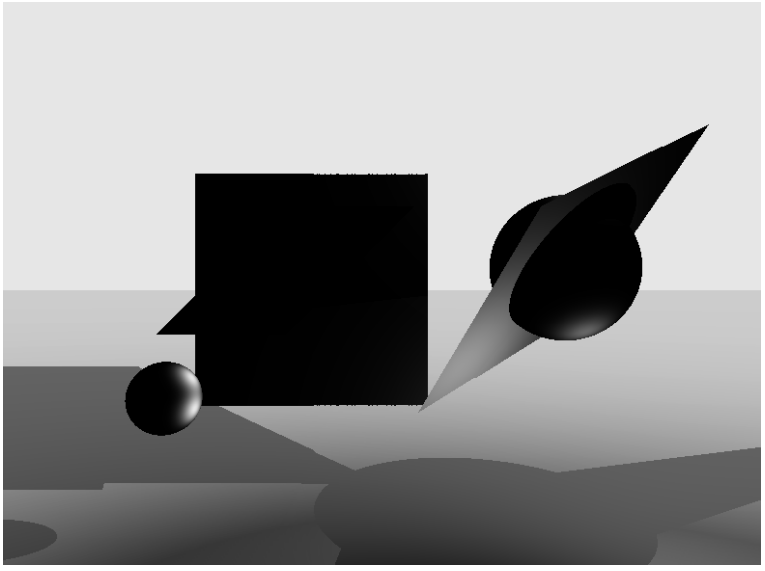


FIGURE 7 – Lumière spéculaire seule

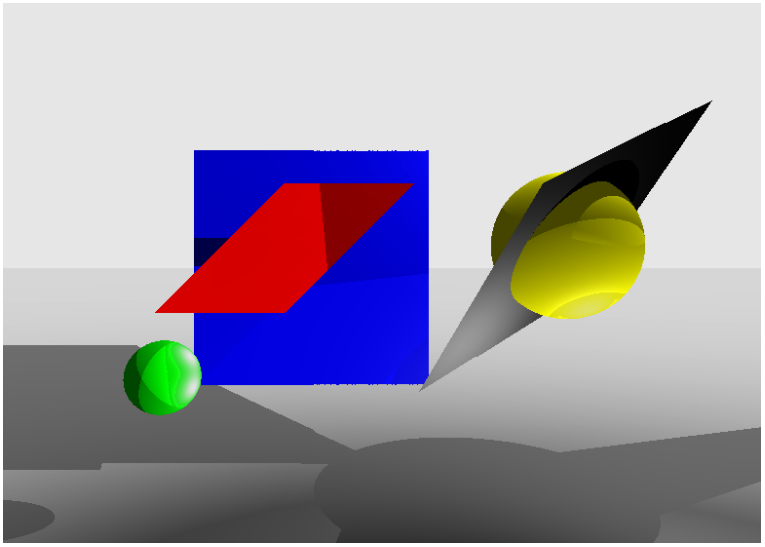
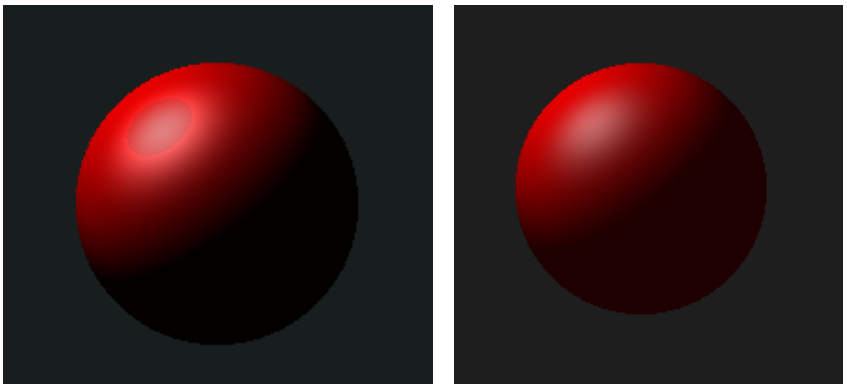


FIGURE 8 – Superposition des trois effets



(a) Problème (b) Résolution

FIGURE 9 – Tâches

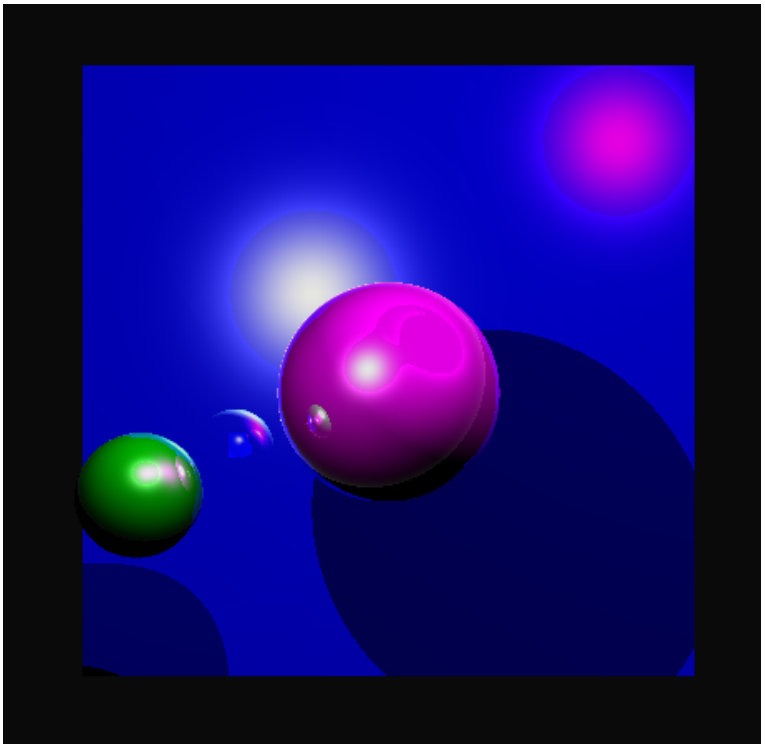


FIGURE 10 – Reflexion

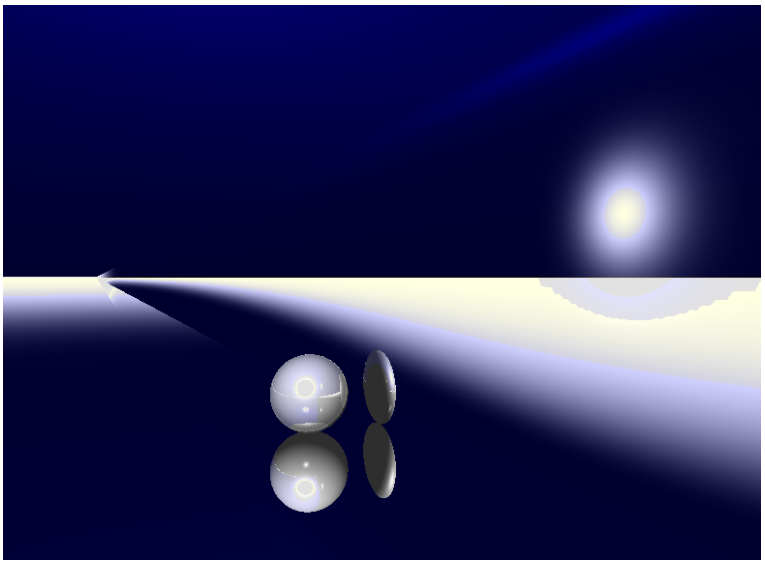


FIGURE 11 – Effet miroir