Projet différencié

Département Informatique
Florent Devin

1 Introduction

Dans ce projet, il s'agit de faire un *raytracer*. Un *raytracer*, ou lancer de rayons, est un logiciel capable de créer une représentation graphique à partir d'une description textuelle. Nous sommes donc dans le domaine de la synthèse d'image. Le lancer de rayons reproduit les phénomènes d'optique étudiés en classe préparatoire, notamment la réflexion ainsi que la réfraction. On peut cependant y intégrer d'autres notions non vues comme les halos, ou autres phénomènes optiques.

Il existe deux familles de lancer de rayons, le *forward* et le *backward*. Le principe du *forward ray tracing* est de se baser sur le principe physique de la lumière. La lumière est émise depuis une source lumineuse. On calcule alors comment cette source éclaire, puis pour chaque objet que la source éclaire, on regarde comment la luminosité est réfléchie. Au final, on ne garde que les rayons issus des différentes sources lumineuses, qui éclairent directement ou indirectement la scène que nous souhaitons visualiser. Le *backward ray tracing* effectue le chemin inverse.

Il existe un *raytracer* très connu dans le monde libre : Pov-Ray. C'est sur ce *raytracer* que nous allons nous baser pour créer le notre. En fait nous allons prendre les mêmes conventions et définitions.

2 Pov-Ray

2.1 Préambule

Il existe trois catégories d'éléments : la caméra ; la(les) source(s) lumineuse(s), les objets. L'espace est repéré par un repère orthonormé XYZ comme le montre la figure 1. Les points sont définis par leur trois coordonnées entre `< >`. Un vecteur sera défini par son vecteur directeur, et sera donc représenté de la même façon qu'un point. Par exemple, `<0,0,0>` définit l'origine du repère, ou le vecteur nul.

Dans Pov-Ray, il est normalement possible de faire des calculs simples et de faire des conversions entre degré et radian.

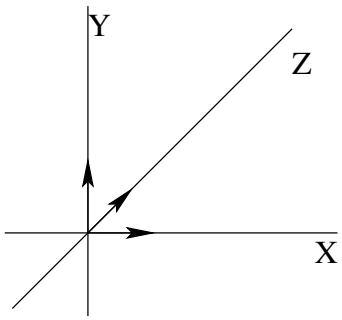


FIGURE 1 – Le répère orthonormé du raytracer

2.2 Qu'est ce qu'une scène ?

Une scène est un fichier texte qui décrit l'ensemble des objets nécessaires au dessin. On retrouvera toujours les mêmes éléments :

1. `#include` : permet d'inclure des fichiers de définition, comme des textures, couleurs, ...
2. `camera` : ceci définit le type de la caméra, sa position, son angle de vue
3. `light_source` : définit la position et la source lumineuse
4. *les objets géométriques, et leurs propriétés* :
   - *forme* : sphere, box, cylinder, cone, torus, plane, ...
   - *operation mathématique* : scale, rotate, translate
   - *texture* : pigment, normal, finish

Par ailleurs, des commentaires peuvent être mis. Ils respectent alors la même règle que les commentaires en C.

Le listing 1 montre une scène très simple, qui représente la figure 2.

Listing 1 – exemple

```
camera{
  location <0,5,-8>
  look_at <0,2,0>
}

light_source{
  <20,50,-50>, color rgb<1,1,1>
}

background{
  color rgb<1,1,1>
}

plane{
  y,0
  pigment{
    color rgb <1,1,0>
```

```
  }
}

sphere{
  <0,2,0>,2
  pigment{
    color rgb <1,0,0>
  }
}
```

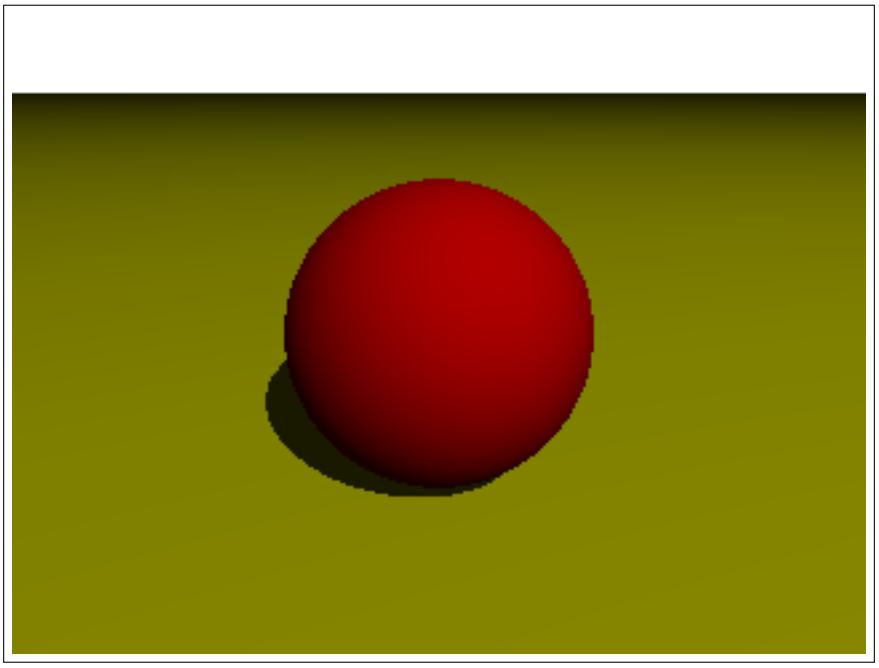


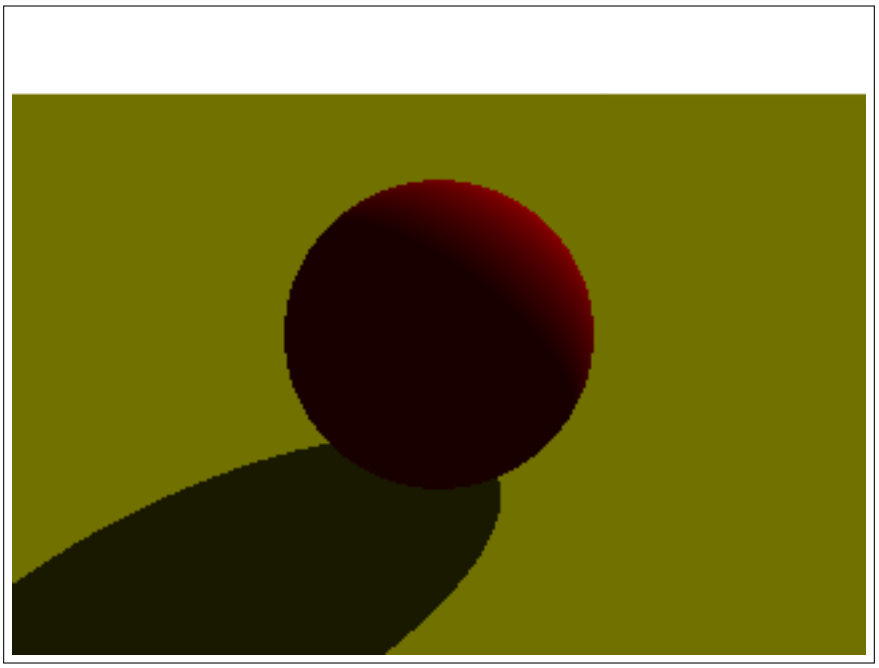
FIGURE 2 – Dessin de la première scène

2.3 Lumières

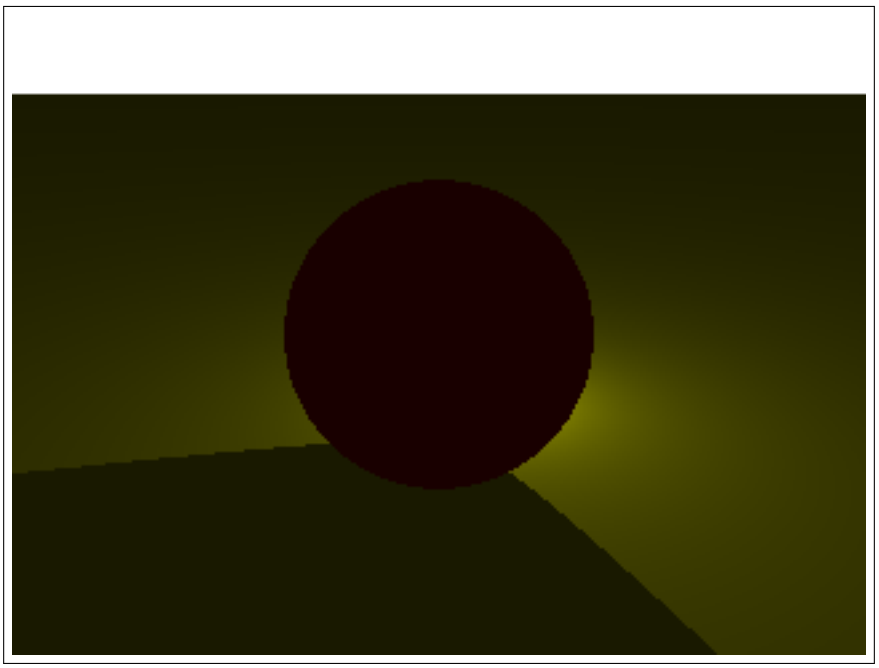
Dans un ray tracer, les lumières sont une part importante de la description. Il existe de nombreuses options pour paramétrer les lumières. Par ailleurs, nous pouvons avoir plusieurs lumières différentes dans une même scène. Dans un premier temps, nous nous intéresserons uniquement aux lumières simple définies par leur position, ainsi que la couleur.

```
// Ici on définit une source lumineuse très lointaine de lumière
// blanche. C'est un peu comme si le soleil venait de naître.
light_source { <200000, 200000, 200000> color rgb <1,1,1> }

// Ici la lumière est beaucoup plus proche, c'est plutôt une ampoule
    !
light_source { <1, 1, 2> color rgb <1, 1, 1> }
```



(a) Lumière infinie



(b) Lumière ponctuelle

FIGURE 3 – Impact de la modification des lumières

Nous envisagerons par la suite d'utiliser d'autres types de lumière, comme des spots, des lumières tamisées, des lumières peu puissantes, ...

2.4 Caméras

La définition d'une caméra est indispensable. Elle représente en fait, la position de l'observateur, ainsi que son angle de vue. Dans Pov-Ray, il est possible de définir plusieurs caméras, et de s'y référer par la suite. Une caméra sera défini, dans un premier temps, par deux attributs : sa position, et sa direction.

```
camera {
  location <0, 1, 10>
  look at <0, 0, 0>
}
```

On pourra envisager de définir des caméras comme dans l'exemple suivant :

```
#declare cam1 =
  camera { location <0, 1, 10>
    look at <0, 0, 0>
}

// utilisation de la camera
camera (cam1)
```

2.5 Couleurs

Il existe plusieurs façons de définir des couleurs, en fonction du mode que l'on souhaite utiliser : rgb, rgbf, rgbt, rgbft.

- rgb : Rouge, Vert, Bleu
- rgbf : rgb, plus intensité du filtre (utile pour fabriquer du verre)
- rgbt : rgb, plus *transmittance* (perte d'intensité)
- rgbft : rgb, plus filtre, plus transmittance

Dans un premier temps, nous nous focaliserons sur le modèle rgb. Les valeurs de chaque composante sont comprises entre 0 et 1. On définira donc `color rgb <0, 0.123, 0.234>`.

2.6 Les objets

2.6.1 Les formes

Il existe plusieurs objets de base, dont voici une rapide présentation :

- `box` : un cube défini par deux points (les deux points opposés)
- `cylinder` : un cylindre défini par deux points (les deux centres), et un rayon
- `torus` : un tore défini par ses rayons internes et externes centrés en l'origine autour de l'axe y
- `sphere` : une sphère définie par son centre et son rayon
- `triangle` : un triangle par ses trois sommets

– `plane` : un plan défini par son vecteur normal et sa distance par rapport à l'origine

D'autres objets existent, mais nous nous limiterons à ceux-ci dans un premier temps.

2.6.2 Les aspects

Nous nous limiterons dans un premier temps uniquement aux pigments définis par une couleur.

2.6.3 Les transformations

Il existe plusieurs transformations :

– *scale* : une homothétie définie par un rapport au centre du repère
– *rotate* : une rotation par rapport à l'origine
– *translate* : une translation relative à la position de l'objet
– *union* : une union de deux objets
– *difference* : la soustraction du deuxième objet sur le premier
– *intersection* : les parties communes à deux objets

Voici un exemple d'objet, et sa visualisation

```
torus { 1.00, 0.25
  rotate <90,90, 135>
  translate <1.25, 2, 0>
  pigment {color rgb <0, 0.3, 0.4>}
}
```

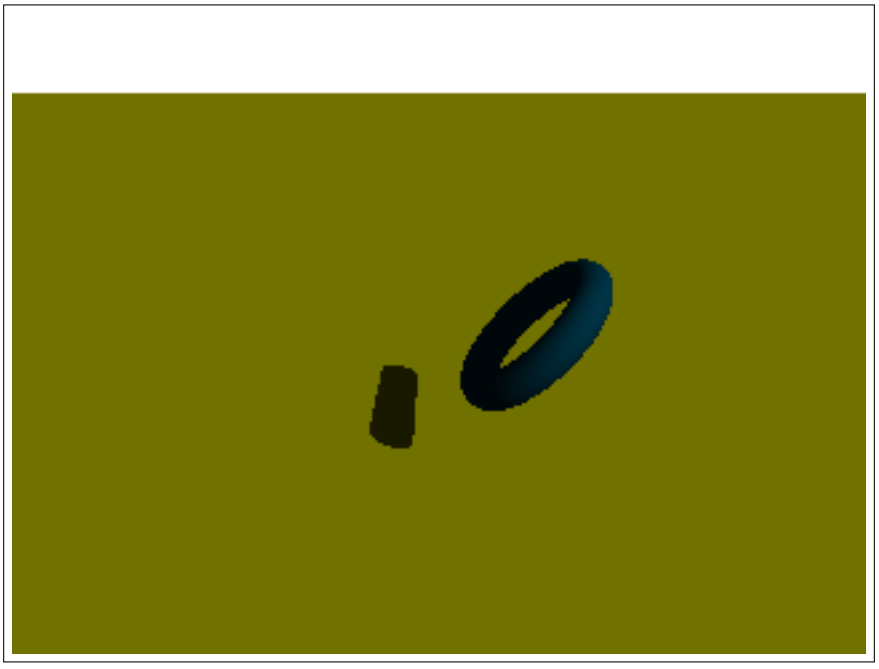


FIGURE 4 – Un tore

3 Travail

3.1 Objectifs

Vous allez donc devoir créer un ray tracer, basé sur le principe du *backward ray tracing*. Celui-ci devra comprendre au minimum toutes les notions exposées. Vous pouvez bien évidemment prendre en compte d'autres notions, si vous le souhaitez. Pour cela, plusieurs étapes seront nécessaires. Afin, de profiter de l'ensemble des scènes déjà écrites par la communauté Pov-Ray, il va falloir que vous écriviez un programme capable de lire et de comprendre une scène. Par ailleurs, il vous faudra trouver comment réaliser un ray tracer, c'est à dire, trouver les équations qui régissent les différents éléments constitutifs de votre ray tracer. Enfin, il vous faudra créer un fichier qui sera le résultat du ray tracing. Pour faciliter la problématique, vous n'aurez pas à afficher directement le résultat, mais générer un fichier au format `PPM`. Il vous suffira alors pour visualiser le résultat de visualiser ce fichier dans un logiciel affichant les images. Avec de la maîtrise, vous pourriez arriver à un rendu comme sur la figure 5.

3.2 Compétences à mettre en place

Vous allez aussi devoir porter une attention particulière à votre développement. Il ne suffit pas que votre programme marche dans un cas pour que cela soit correct. Vous devez tester toutes les situations, afin de fournir un programme correct. Vous devrez aussi porter une attention particulière à vos méthodes de développement.

Par ailleurs, la notation tiendra compte de la performance de chaque module. Il vous faudra donc utiliser des outils de *profiling* pour vous permettre de détecter les fuites mémoires, ainsi que les problèmes de performance liés soit à une mauvaise conception, à un mauvais choix technique, ou une mauvaise traduction de l'algorithme initial.

La présence en séance de projet est obligatoire. Toute absence entraînera une pénalité. Le projet étant ambitieux, il vous appartient aussi de travailler en dehors du temps de présence.

FIGURE 5 – Un joli dessin