

# Lancer de Rayon

## Rapport du projet de synthèse d'image M1

---

Hadjer Djerroumi & Jean-Manuel Erialc

June 7, 2020

## 1 AVANT COMPIRATION

### 1.1 Bibliothèques utilisées

En plus de la librairie standard du langage C, nous avons utilisé:

- *libxml2* permettant de parser un fichier xml. Elle est incluse dans les principales distributions unix. Il est possible de l'installer via la commande: *sudo apt-get install libxml2 libxml2-dev*
- *libg3x* de Monsieur Eric Incerti pour la création de la fenêtre d'interaction.

### 1.2 Description du fichier d'entrée

Le programme nécessite un fichier décrivant la caméra et la scène. C'est un fichier de type XML se trouvant dans le répertoire *input*.

#### 1.2.1 La caméra

Elle possède trois champs:

- *resolution*: taille de l'image de sortie (2 champs).
- *screen*: taille de l'écran (2 champs).
- *eye*: représente un point (3 champs).

#### 1.2.2 Objets de la scène

Le programme traite 5 types d'objets.

- **Sphère**: elle est définie par son centre et son rayon.
- **Cube**: le cube doit être défini par ses six faces, mais vu la difficulté en pratique de trouver des instances quelconques, nous nous sommes limités à des cubes parallèles aux axes (*définis par un centre et un rayon*).
- **Cylindre**: défini par deux points *P* et *Q* et son rayon. La hauteur est le segment *[PQ]* et l'axe est la droite *(PQ)*.

- **Rectangle:** nous l'avons défini par:
  - centre.
  - rayon: égale à la moitié de la diagonale (la plus grande).
  - deux vecteurs distincts  $u$  et  $v$  définissant le plan du rectangle et dont l'angle nous définit le reste des angles.
- **Triangle:** Défini par ses trois points.

En plus de ces données, chaque objet possède une couleur *RGB*. Les coefficients de réflexion diffuse, spéculaire et la brillance sont fixés suivant ce tableau:

| Object    | r_diffuse   | r_spéculaire | brillance |
|-----------|-------------|--------------|-----------|
| Sphère    | couleur/255 | 0.6          | 60        |
| Cylindre  | couleur/255 | 0.8          | 50        |
| Cube      | couleur/255 | 1.0          | 200       |
| Rectangle | couleur/255 | 1.0          | 80        |
| Triangle  | couleur/255 | 1.0          | 100       |

Table 1.1: table de coefficients.

### 1.2.3 Sources Lumineuses

Les sources lumineuses dans ce projet sont ponctuelles. Chaque source est définie par un point et une intensité d'ordre 5 à 10 millions.

## 2 COMPILEMENT ET EXÉCUTION

Dans le répertoire racine du projet, la compilation se fait avec la commande *make*, et l'exécution comme indiqué sur l'énoncé du projet à savoir:

- *niveau 1*: ./lray -n 1 -i <input file> -o <output file> [-t]
- *niveau 2*: ./lray -n 2 -i <input file> [-o <output file>] [-t]
- *niveau 3*: ./lray -n 3 -ps <pixel sampling> -i <input file> [-o <output file>] [-t]

L'option *-t* en fin de ligne correspond au timing du calcul de la scène.

## 3 INTÉRACTION UTILISATEUR

Dans le niveau 2, l'utilisateur pourra:

- déplacer la caméra vers la gauche ou vers la droite avec les touches **K** et **M** respectivement (des rotations autour de l'axe oy).
- déplacer la caméra vers le haut ou vers le bas avec les touches **O** et **L** respectivement (des rotations autour de l'axe ox).
- zoomer ou dézoomer avec les touches **P** et **I** respectivement (des translations).
- activer ou désactiver le rendu progressif (régulier avec 9 rayons) avec un bouton sur la fenêtre.
- ajouter (*resp.* enlever) des réflexions spéculaires pures avec la touche **+/-** (*resp.* **-**).
- enfin si un rendu lui plaît, il pourra le sauvegarder (dans le répertoire *output*) avec la touche **S**.

Au niveau 3, les options restent valables mais l'intéraction n'est pas très pratique car le *pixel sampling* est très coûteux.

#### 4 EXEMPLES DE RENDU

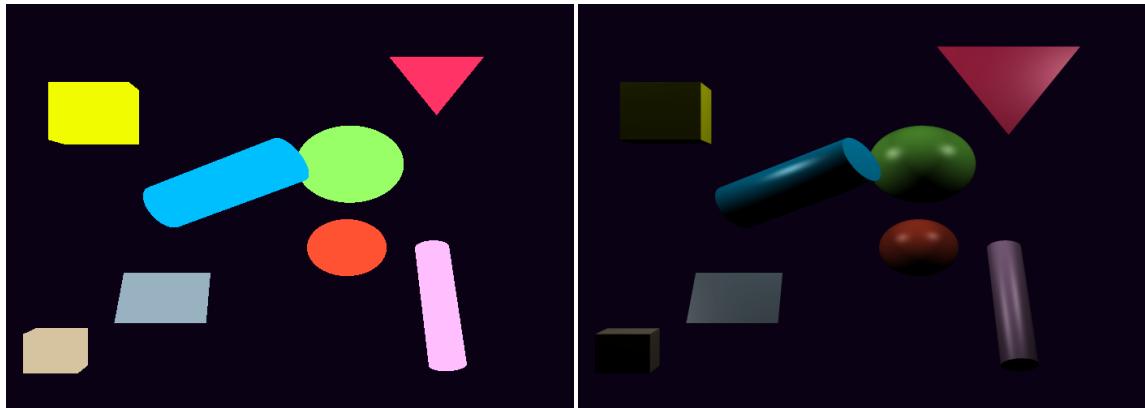


Figure 4.1: Niveau 1 à gauche et niveau 2 à droite

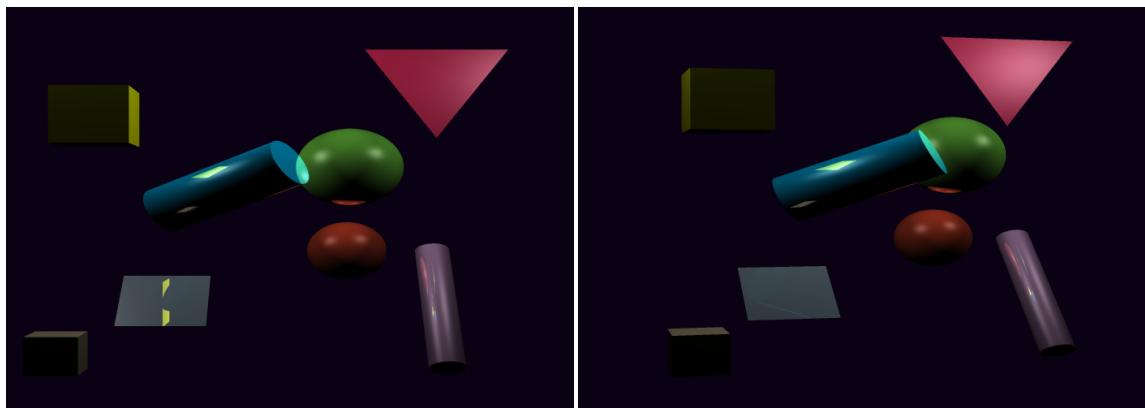


Figure 4.2: Niveau 2 avec reflets et avec déplacement de la caméra.

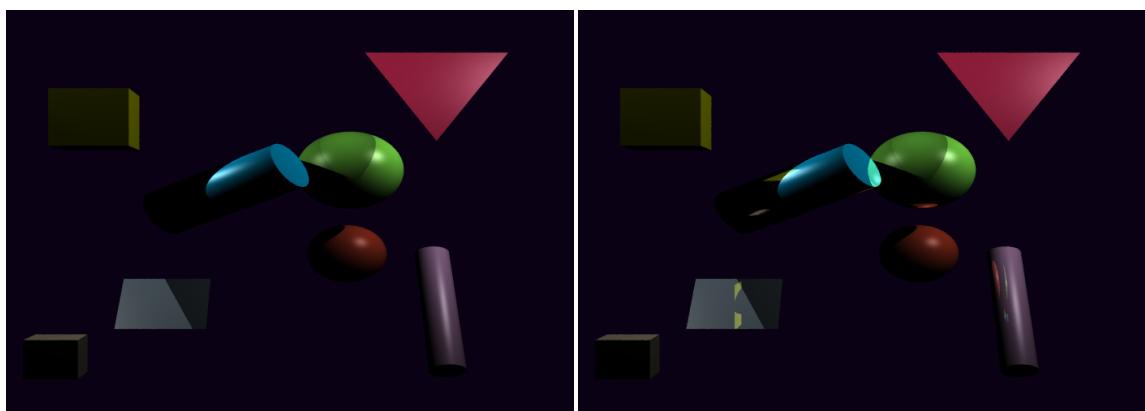


Figure 4.3: Niveau 3, 2 sources lumineuses, avec et sans reflets.

## 5 EXTENSIONS

*Extensions des formats d'entrée et de sortie:*

Le fichier d'entrée est au format XML plutôt qu'au format TXT, et la scène est exportée au format PNG plutôt qu'au format PPM.

*Des algorithmes de création procéduraux de scènes complexes:*

Nous avons implémenté un générateur de scènes complexes. L'utilisateur pourra choisir le nombre d'objets ainsi que leurs types. Dans le répertoire racine du projet, il se compile par : `make generator`, et s'exécute par :

```
./bin/generator <w> <h> [-p <type of objects eg. sphere,rectangle>] -n <number of objects>  
-o <output_file.xml> [-t]
```

où  $w * h$  est la résolution souhaitée.

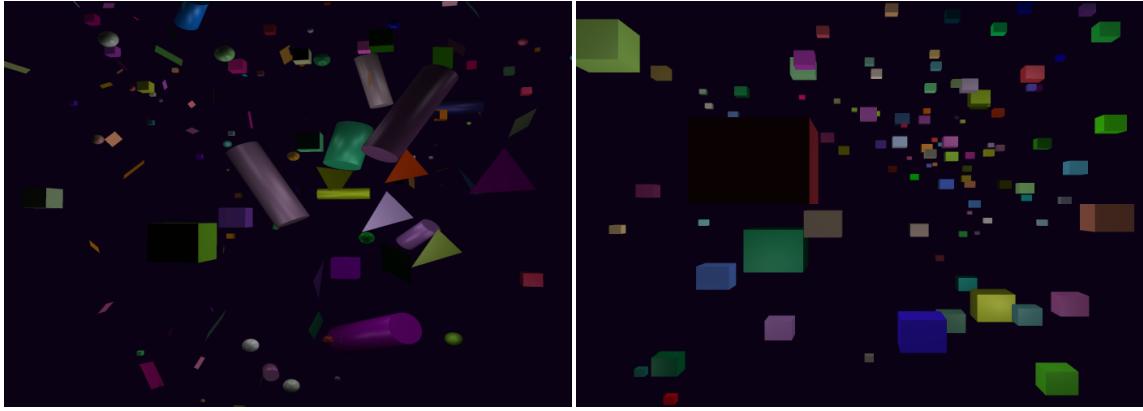


Figure 5.1: Deux scènes aléatoires, une avec tous les types et l'autre avec des cubes.

## 6 EXEMPLES DE TIMING

Grâce à l'option `-t` de la commande d'exécution, nous avons les résultats suivants:

| Scène    | 8 objets | 50 objects | 150 objets |
|----------|----------|------------|------------|
| Niveau 1 | 0.189    | 1.012      | 3.595      |
| Niveau 2 | 0.221    | 1.085      | 3.664      |

Table 6.1: table du timing en secondes de 3 scènes (niveaux 1 & 2).

| pixel sampling    | ps = 1 | ps = 10 | ps = 20 |
|-------------------|--------|---------|---------|
| temps d'exécution | 0.32   | 2.70    | 5.28    |

Table 6.2: Exemple du timing en secondes d'une scène de 8 objets (niveau 3).