Raphaël GOSSET / Yanis LAASSABI

Rapport de projet de synthèse d'image

Contexte et objectifs

Ce projet de synthèse d'image a été réalisé dans le cadre du module **LIFIMAGE**. L'objectif était de construire un moteur de **lancer de rayons (ray tracing)** en C++ capable d'afficher une scène 3D composée d'objets variés (plans, sphères, maillages .obj), avec des fonctionnalités graphiques avancées :

- Flou de profondeur (depth of field)
- Éclairage Phong (diffus + spéculaire)
- Reflets miroir simples
- Importation d'objets Blender (.obj)
- Accélération via un BVH (Bounding Volume Hierarchy)
- Post-traitement (flou gaussien, filtre bilatéral)

Nous avons également refactoré le code de manière propre en suivant une logique modulaire.

Fonctionnalités implémentées

1. Lancer de rayons

Le moteur fonctionne en traçant plusieurs rayons par pixel, ce qui permet un anti-crénelage naturel. Chaque rayon est lancé depuis une caméra décalée (lentille) vers un point focal.

2. Primitives géométriques

- **Sphères** : définies par centre et rayon, avec test d'intersection analytique.
- **Plans** : sol avec motif damier, intersection par projection.
- **Triangles** : chargement de mesh .obj en triplets de sommets, avec test de barycentrie.

3. Importation d'objets Blender

Les fichiers `.obj` sont chargés avec `read_positions()` (sans les matériaux). Pour chaque triplet de points, un triangle est ajouté à la scène.

Un **offset** (vecteur de décalage) est appliqué pour positionner correctement l'objet dans la scène. On calcule également la **normale** du triangle, et si elle pointe vers le bas, on inverse deux sommets pour corriger le winding.

> **Difficulté rencontrée :** le rendu était initialement incohérent, les triangles étaient affichés dans tous les sens. Le problème venait du winding incorrect, corrigé par inversion conditionnelle des sommets.

Raphaël GOSSET / Yanis LAASSABI

4. Éclairage de type Phong
L'éclairage repose sur le modèle **Phong** :

- **Composante diffuse** : produit scalaire entre la normale et la direction de la lumière.
- **Composante spéculaire** : comparaison entre la direction réfléchie et la direction du rayon, avec un exposant de brillance (32).

> **Difficulté rencontrée :** pendant le développement, nous avons rencontré un bug où l'image rendue était noire. Cela était lié à une erreur dans le produit scalaire entre la normale et la lumière, ce qui réduisait à tort l'opacité.

5. Objets miroirs
Pour simuler des réflexions simples, si un objet est marqué `mirror`, on relance un rayon réfléchi, et on ajoute sa contribution à la couleur finale (atténuée).

6. Sky Dome (ciel hémisphérique)
Si aucun objet n'est touché, un échantillonnage de directions est fait dans un hémisphère orienté vers le haut (zenith), avec une intensité décroissante selon l'angle.

7. Flou de profondeur (depth of field)
On simule une lentille avec un **disk sampling** : chaque rayon est lancé depuis un point aléatoire autour de la caméra (lentille), convergeant vers un point focal. Cela permet de flouter les objets hors focus.

8. BVH (Bounding Volume Hierarchy)
Pour accélérer les intersections rayon-triangle, un **BVH** est construit :

- Tri des triangles selon un axe (x, y, z) en fonction de la profondeur.
- Division récursive jusqu'à un nombre minimal de triangles.
- À chaque noeud, on stocke une AABB englobante.

Des statistiques sont affichées :

- Nombre de noeuds
- Nombre de feuilles
- Profondeur maximale

9. Refactorisation du code
Le code a été divisé proprement en fonctions :

- `load_scene()` : charge les objets et construit le BVH
- `render_scene()` : boucle principale de rendu
- `trace_ray()` : définit la couleur pour un rayon donné
- `compute_lighting()` : calcule éclairage Phong

- `delete_bvh()` : libère la mémoire du BVH

Les structures sont bien nommées et concises (`Hit`, `Triangle`, `Plan`, `Sphere`).

10. Filtres OpenCV (bonus)
Après le rendu, on convertit l'image en `cv::Mat` (OpenCV) pour appliquer :

- **Un flou gaussien** : lisse les contours
- **Un filtre bilatéral** : floute en préservant les bords

Cela permet d'améliorer la lisibilité de l'image sans modifier le rendu brut du ray tracing.

Démarche de développement

Voici les étapes que nous avons suivies dans l'ordre :

1. Intersection rayon-sphère, rayon-plan
2. Mise en place du damier
3. Lumières directionnelles
4. Support des objets miroir
5. Éclairage Phong
6. Lecture du .obj avec `read_positions`
7. Correction du winding (normales incohérentes)
8. Construction du BVH
9. Refactorisation totale du code
10. Ajout des effets OpenCV (bonus)

Conclusion

Ce projet nous a permis de mettre en pratique l'ensemble des techniques fondamentales du ray tracing :

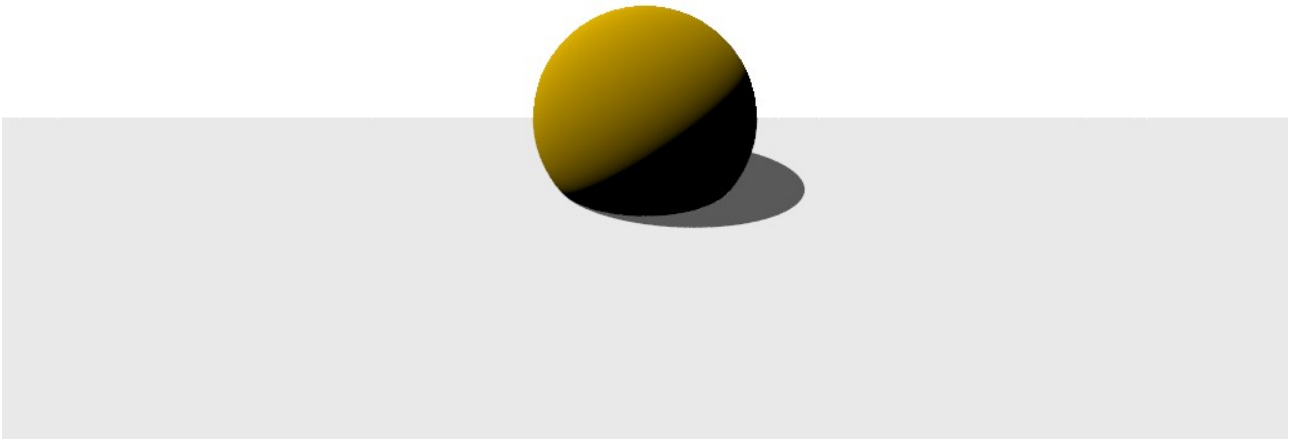
- Intersections géométriques
- Modèles de lumière (Phong)
- Structures d'accélération (BVH)
- Post-traitement graphique

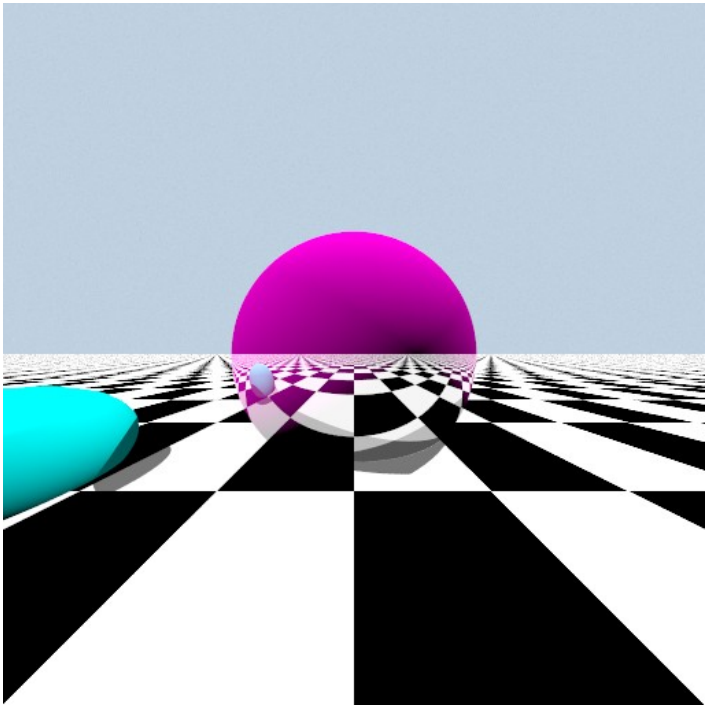
Les principales difficultés ont concerné :

- Le calcul des normales lors de l'import Blender

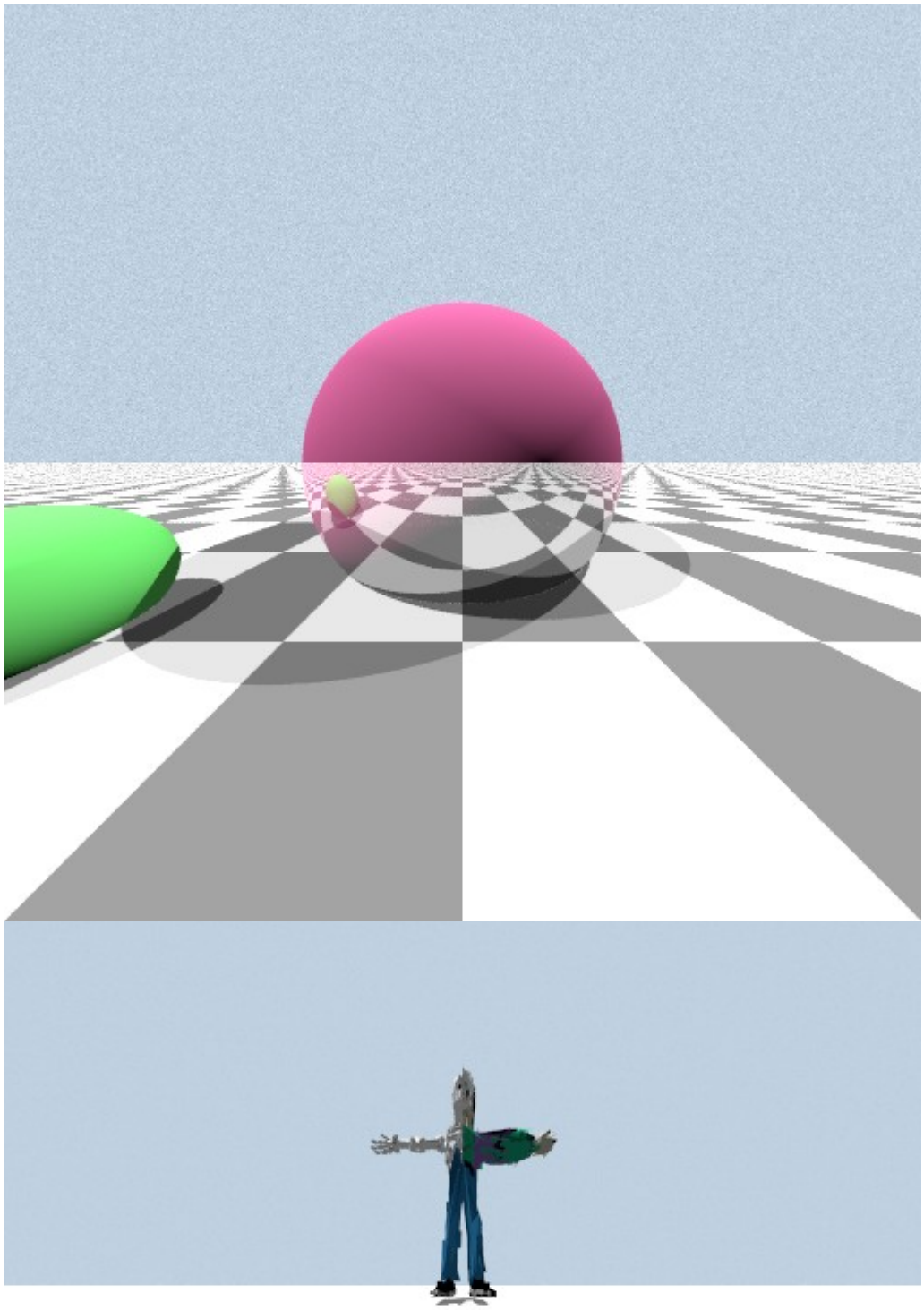
Raphaël GOSSET / Yanis LAASSABI

- Le bug d'opacité causé par un produit scalaire mal géré
- Le réglage de la caméra pour obtenir une vue artistique et compréhensible

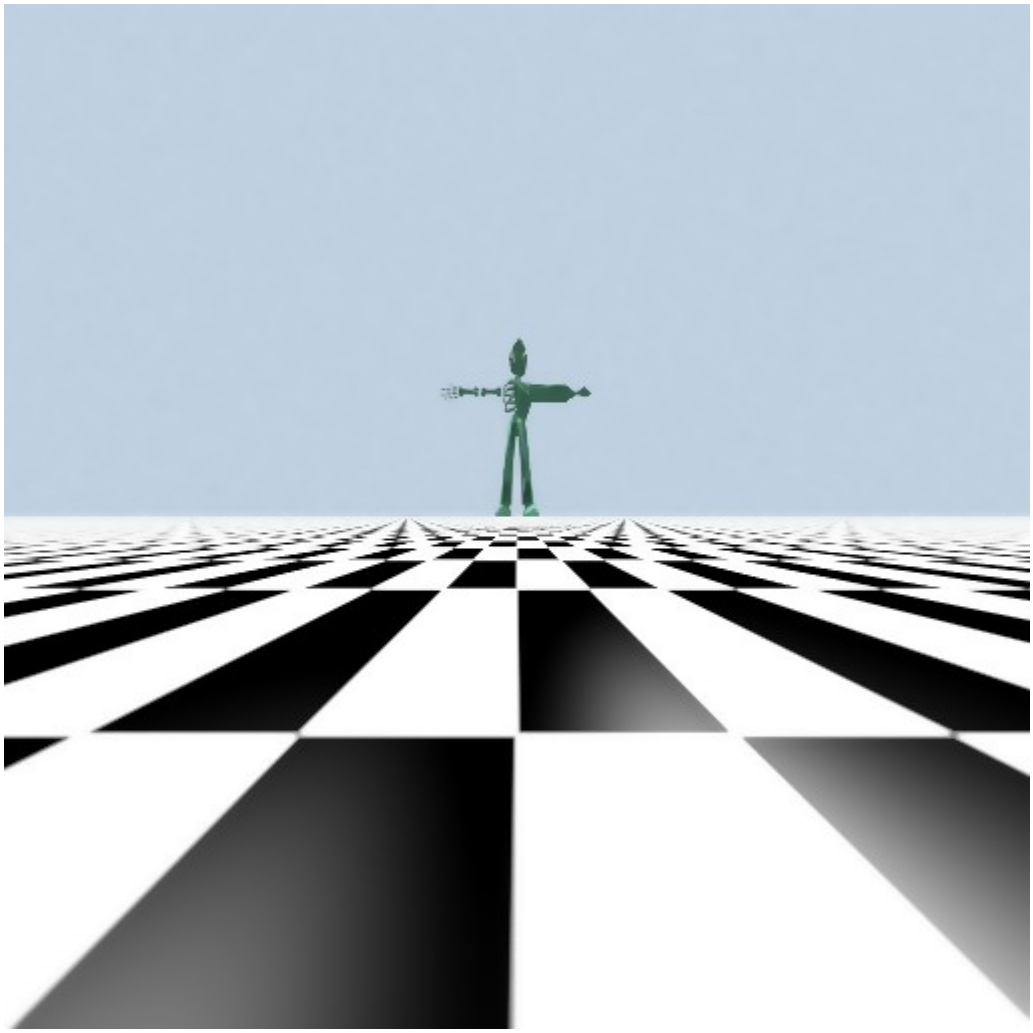




Raphaël GOSSET / Yanis LAASSABI



Raphaël GOSSET / Yanis LAASSABI



Raphaël GOSSET / Yanis LAASSABI