Rapport d'analyse du projet de Technologies Objets

Maxime Arthaud Korantin Auguste Martin Carton

24 Mai 2013

1 Introduction

Ce projet consiste en la création d'une interface utilisateur permettant de gérer une scène 3D et à la réalisation d'un moteur de rendu par lancé de rayons.

2 Démarche

Nous avons commencé par la liste des choses à faire. Nous avons décidé de séparer l'interface graphique du moteur de rendu. L'interface graphique permettra de générer un fichier contenant toutes les informations de la scène nécessaires à la génération de l'image, qui sera faite par un second programme utilisable en ligne de commande, ce fichier sera simple et pourra être écrit à la main.

3 Tâches effectuées

Nous avons commencé à réfléchir à l'organisation des classes, à l'interface utilisateur et au format de fichier.

3.1 Interface utilisateur

La figure 1 présente l'interface graphique telle que nous envisageons de la faire.

Les différents objets (caméra, sphères, cubes et plans) peuvent être y ajoutés selon leurs différentes propriétés optiques et supprimés. Un résumé des objets créés est affiché, on peut directement lancer la génération de l'image, ou enregistrer le fichier.

3.2 Format de fichier

Le fichier sera un fichier texte où chaque objet sera représenté sur une ligne. Chaque ligne sera de la forme `Nom(param1=valeur1, param2=valeur2, .., paramN=valeurN)` où `Nom` est le nom de l'objet à créer (Camera, Sphere, Cube ou Plane), et les paramètres et valeurs sont les données qui caractérisent l'objet, par exemple le rayon et le centre pour la Sphere, ou encore la réflectance.

Pour simplifier l'écriture manuelle du fichier, certains paramètres ont une valeur par défaut, comme l'indice de réfraction.

3.3 Moteur de rendu

Nous avons décidé de structurer notre programme de la manière suivante : tout d'abord, la classe `FileReader` va se charger de parseur le fichier décrivant la scène.

La méthode `computeColor` sera chargée de donner la couleur d'un rayon arrivant sur l'objet (et appellera récursivement cette même méthode pour d'autres objets, dans la plupart des cas). Une seconde méthode, `distance`, est chargée de donner la distance parcourue le long du rayon passé en paramètre pour arriver jusqu'à l'objet.

Un objet pourra soit être un `Mesh`, composé de plusieurs triangles (ce sera le cas d'un cube), soit être un objet possédant tous les caractéristiques optiques nécessaires.

Il sera donc très facile de générer notre image : il suffira, pour chaque rayon commençant par le point de vue de l'observateur et passant par un pixel de l'écran, de trouver quel est le premier objet qu'il intersecte grâce à la méthode `distance`, et d'utiliser la méthode `computeColor` pour calculer la couleur de ce point de l'objet.

Les classes `javax.imageio.spi.ImageWriterSpi` et `javax.imageio.ImageWriter` permettent de gérer les images au format PPM en utilisant les interfaces standards de java, ce qui nous permettra de pouvoir exporter aussi bien en PPM qu'en PNG, format supporté par défaut par java.

La figure 2 présente les relations entre les différentes classes que nous envisageons d'utiliser.

4 Démarche et répartition des tâches

Nous avons décidé de développer l'interface graphique en dernier. Nous comptons faire des tests unitaires pour les classes `FileReader` et `PPMWriter`. Nous ferons des tests d'intégrations en maintenant un ensemble de fichiers de scène qui couvrent l'ensemble des fonctionnalités du programme.

Nous avons déjà répartis quelques tâches :

- Maxime : travaillera sur la GUI ;
- Martin : travaillera sur le parseur et l'enregistrement d'image ;
- Korantin : rédaction des rapports et comptes rendus, et travail sur le moteur de rendu.

Nous allons tous participer à l'implémentation du moteur de rendu.

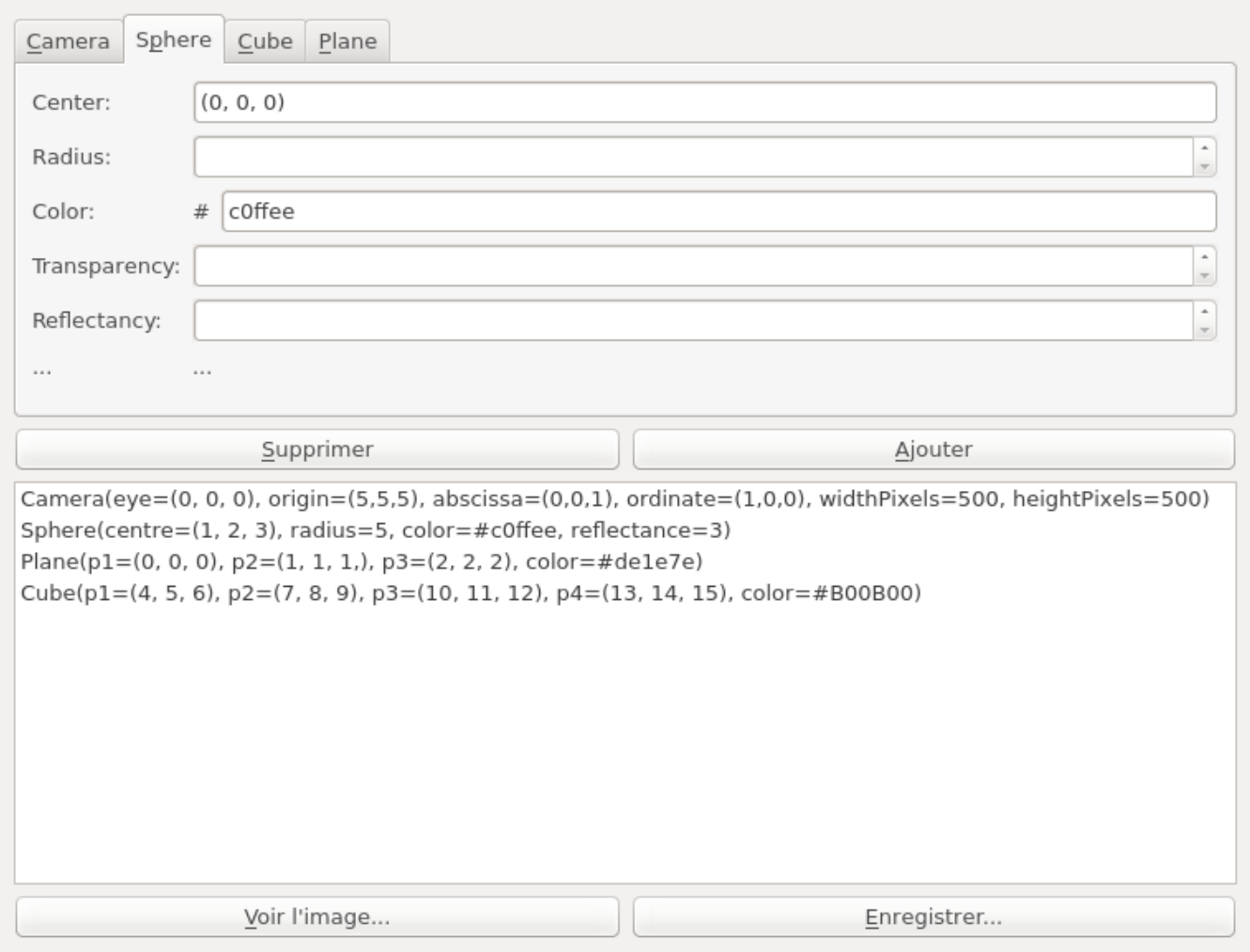

FIGURE 1 – Interface graphique

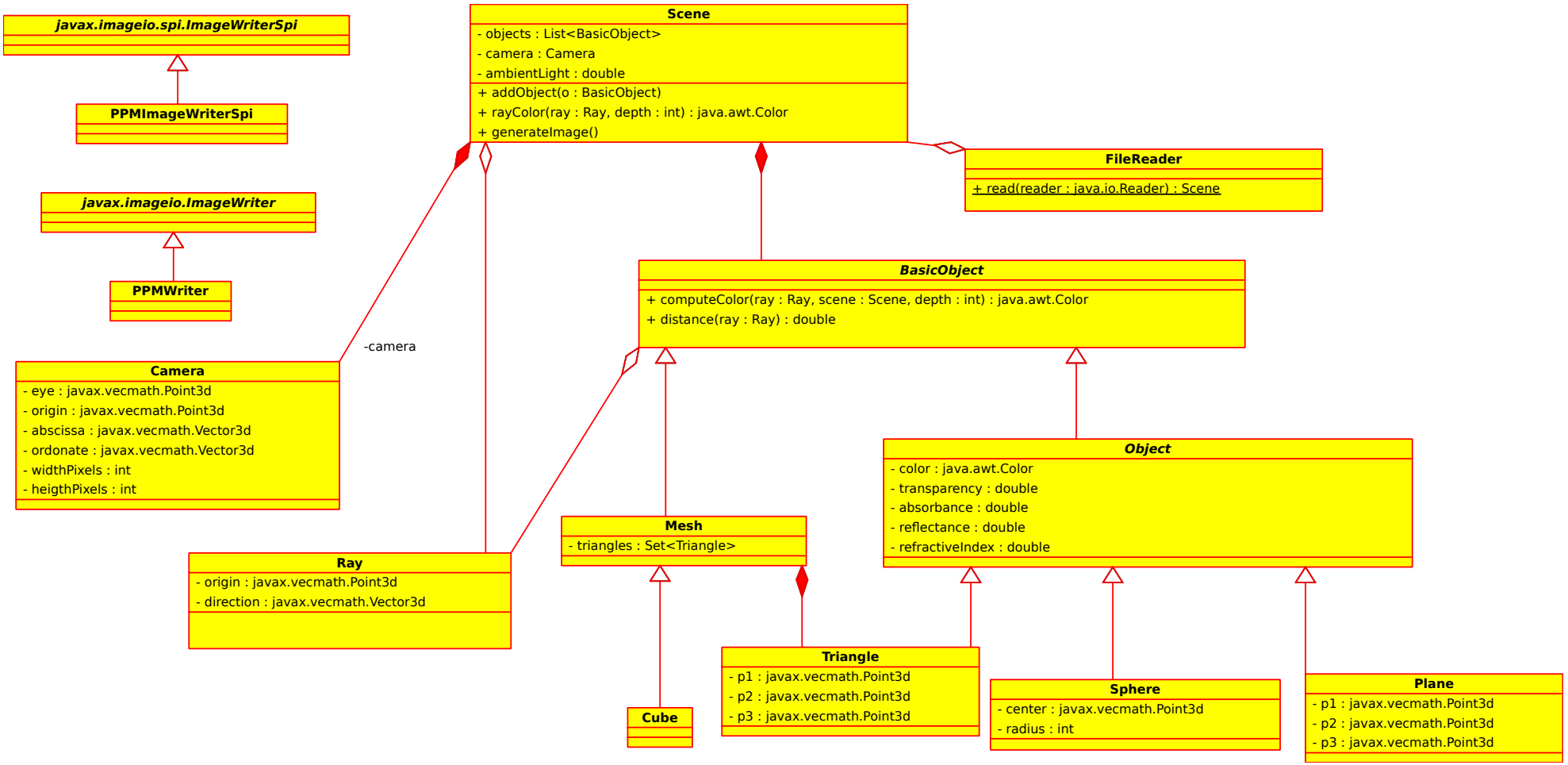

FIGURE 2 – Diagramme UML