Projet de Java – Partie 2

Licence 3 – Informatique

Rémi Forax, Pierre Peterlongo, Benoit Olivieri
(forax@univ-mlv.fr, pierre.peterlongo@univ-mlv.fr, olivieri@univ-mlv.fr)

Description du projet

Le programme RayOfLight est un petit programme de Raytracing (lancé de rayon) permettant facilement de faire de la synthèse d'image simple.
Il existe déjà beaucoup de *raytracers* qui marchent très bien et qui font beaucoup de choses, le but n'est pas d'implanter toutes les fonctionnalités d'un *raytracer* classique mais un sous-ensemble cohérent et qui fonctionne.

Calendrier

Ce projet est à faire par binômes (cela veut dire deux personnes pas trois ni une).
Le rendu du projet est découpé en deux parties :

1. Pré-rapport indiquant l'architecture objet et tests d'utilisation.
   Date de rendu : le 31 mars, avant minuit.
2. Rendu du projet en lui-même.
   Date de rendu : le 11 mai, avant minuit.

   Chaque rendu s'effectuera par mail sous forme d'une pièce jointe au format ZIP contenant l'ensemble des programmes et documents. Le mail devra être envoyé avant minuit aux trois adresses données au début de ce document.

   Le fichier s'appelera nom1_nom2.zip avec nom1 et nom2 les nom des binomes dans l'ordre alphabétique.

Correction du premier rendu

Répondre aux questions :

Comment sont codées les transformations sur les objets ?

En utilisant une matrice 4x4 en coordonées homogène représentant la composition des différentes transformations effectuées sur les objets. (voir la classe fr.umlv.rol.Matrix)

Comment l'algorithme de raytracing est codé en terme d'appels de méthodes entre les différents objets ?

On appelle sur la scène (fr.umlv.rol.Scene) la méthode render(Camera camera,IlluminationModel<T> model) qui renvoie l'image générée. Cette méthode demande à la camera de créer un rayon pour chaque point de l'écran en utilisant createRay(int pixelX,int pixelY).

Puis pour chaque objet de la scène (de type fr.umlv.rol.SolidObject), on calcule sa distance par rapport au rayon lancé (ce en passant le rayon dans le repère de l'objet) en utilisant la méthode intersect(Ray ray).
On récupère l'objet qui est le plus proche s'il existe et l'on calcule la couleur au point d'impact du rayon avec la méthode processSolidColor(Ray ray,SolidObject object,float distance,Light[] lights,T modelData) sur le modèle d'illumination (fr.umlv.rol.IlluminationModel).

Comment faire pour représenter les différents objets de la scène (boule et plan) ?

On utilise une interface (fr.umlv.rol.SolidObject) représentant un type commun aux types shpère (fr.umlv.rol.Sphere) et plan (fr.umlv.rol.Plane).

Qu'implique le format du fichier XML sur les objets ?

Il est possible :

d'appliquer une couleur sur une lumière ou sur un objet donc ceux-ci doivent avoir une interface commune (fr.umlv.rol.ColoredObject)

les objet possèdent un material (fr.umlv.rol.Material)

d'appliquer une transformation sur un objet ou une caméra, donc type commun (fr.umlv.rol.MovingObject)

d'utiliser un paramètre spécifique au modèle de whitted (depth), le modèle d'illumination sera donc paramétré par les données qu'il peut prendre en entrée (Void pour tous le monde sauf Integer pour whitted)

Comment faire pour utiliser un modèle d'illumination en fonction de ce qui est demandé dans le fichier XML ?

Il faut associer à une chaine de caractère le modèle correspondant, par exemple en utilisant une association (java.util.Map)

Diagramme des classes

L'architecture est composée de 6 classes et 7 interfaces :

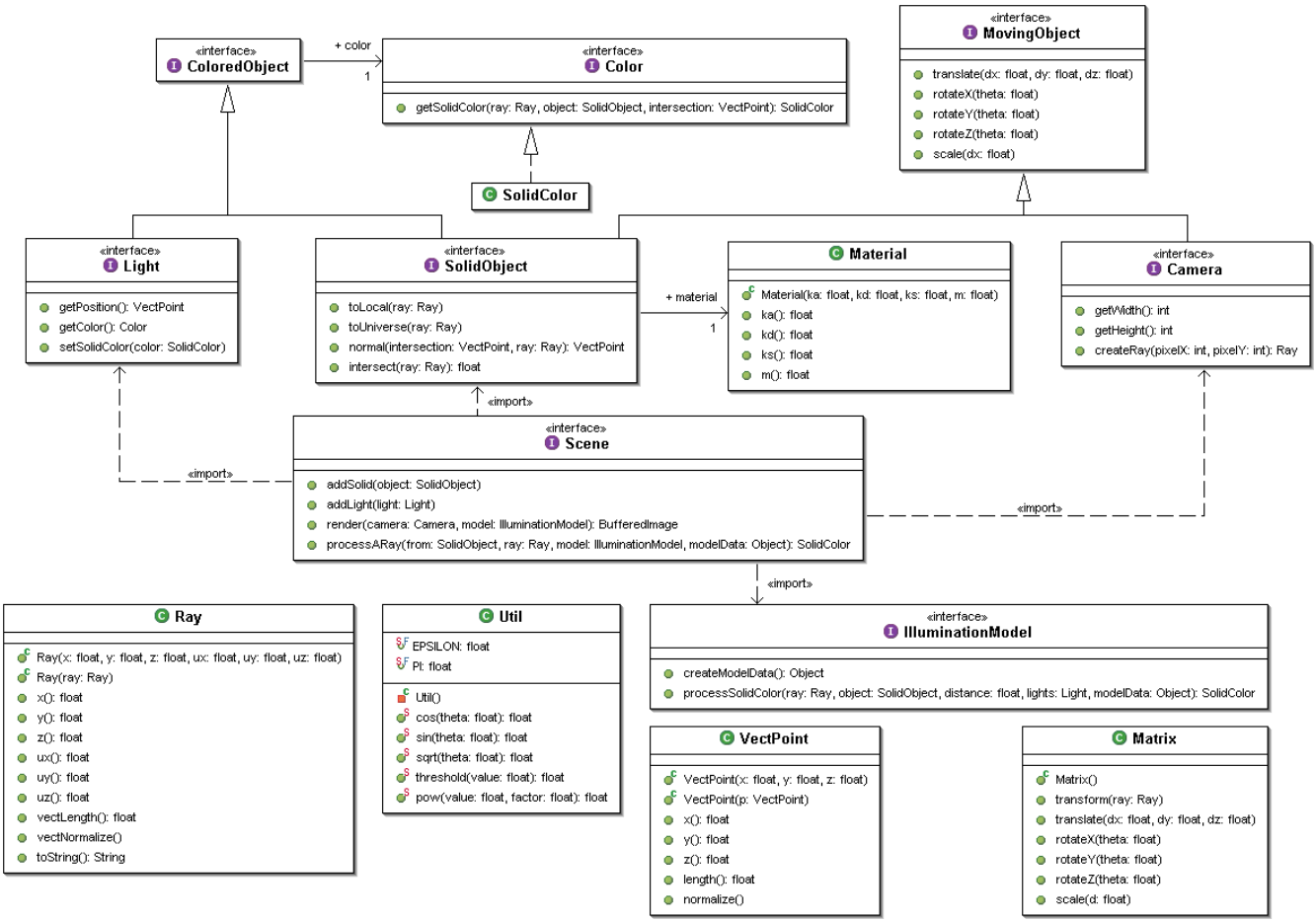
fr.umlv.rol.ColoredObject correspond à un objet coloré (lumière ou objet de l'espace de la scène). fr.umlv.rol.Color correspond à l'interface de gestion des couleurs.
fr.umlv.rol.SolidColor correspond au couleur RGB utilisant des composantes flottantes.
Le fait d'avoir une interface Color permet assez facilement de rajouter un mécasnisme de texture 3D ou plaquée.

fr.umlv.rol.MovingObject correspond aux objets pouvant subir des transformations donc les objets de la scène et les caméra (fr.umlv.rol.Camera).

fr.umlv.rol.Material contient tous les coefficients nécessaires aux modèles d'illumination (fr.umlv.rol.IlluminationModel).

fr.umlv.rol.Ray correspond à un rayon, fr.umlv.rol.Matrix à une matrice de transformation,

fr.umlv.rol.VectPoint à un vecteur ou un point 3D.
fr.umlv.rol.Util aux opérations nécessaires pour le calcul sur des floats.



Détail du second rendu

Le second rendu devra impérativement être effectué par les mêmes binome que le précédent rendu.

Voici le nom des répertoires et fichiers qui doivent être contenus dans l'archive zip.

1. un fichier **readme.txt** indiquant comment compiler et exécuter le programme, où se trouve la doc, *etc.*
2. un fichier Ant **build.xml** permettant de compiler les sources du programme, et de créer le jar exécutable **rayoflight.jar**.
3. un répertoire **src** contenant l'ensemble des sources (.java) sous forme de plusieurs paquetages. Les interfaces et classes contenues dans fr.umlv.rol ne doivent pas être modifiées.
4. un répertoire **classes** contenant l'ensemble des classes (.class) correspondant aux sources sous forme de plusieurs paquetages.
5. un répertoire **lib** contenant l'ensemble des JARs correspondant aux bibliothèques externes utilisées.
6. un répertoire **bin** contenant deux fichiers, **rayoflight.sh**, script shell démarrant le logiciel sous Linux (en fait Unix) et **rayoflight.bat** démarrant le logiciel sous Windows.
7. un répertoire **docs** contenant deux documents au format PDF :

1. La documentation utilisateur **user.pdf** contenant en plus des informations classiques (comment compiler, exécuter, etc.) une description de l'application, et comment l'utiliser
2. La documentation développeur **dev.pdf** contenant :
   Une description détaillé et au niveau de chaque classe que vous avez implantée.
   Evitez les copier/coller de la javadoc, essayer plutot de les décrires par leur responsabilité et dans un ordre correspondant à algorithme de raytracing.
   Une liste des bugs connus (s'il y en a) détaillant le scénario permettant de générer le bug ainsi que la raison pour laquelle celui-ci se produit.

En plus des deux documents, le répertoire **docs** devra contenir un sous-répertoire **api** contenant la documentation complète du logiciel au format javadoc.

Bug connu

Dans l'implantation fourni des matrices (fr.umlv.rol.Matrix) il y a un bug au niveau de la gestion des rotations. Un bonus de +2 est prévu pour chaque binome qui débugguera se code.