30 avril 2020

Projet rendu 3D

Travail Personnel approfondi

Marguerite Bauchez 21803320 Olivier Cocquerez 21803239
Paul Lebranchu 21403460 Raphaelle Lemaire 21802756

L2 informatique
Groupe 3A
2019-2020

Table des matières

Introduction

Nous avons choisis le projet de rendu 3D par lancer de rayon. Notre objectif est de réaliser un moteur de rendu 3D qui utilise la technique de lancer de rayons (technique permettant de réaliser des rendu 3D réaliste en se basant sur les lois de la lumière). Pour cela,nous devons récupérer les informations issus d'un document POV-Ray à l'aide d'un parser et créé un moteur de rendu 3D qui générera les images au format png. Nous avons utiliser le langage java pour rédiger notre code et nous avons créé une zone de travail sur SVN pour faciliter le travail de groupe.

1 Prise en main du projet

1.1 POV-Ray

POV-Ray signifie Persistence Of Vision Raytracer, il s'agit d'un logiciel de lancer de rayon (c'est à dire un logiciel capable de générer des images en 3D en se basant sur la technique des lancer de rayons). Sa première version est sorti en juillet 1991 et sa dernière mise à jour date de novembre 2013 (version 3.7).

Pour générer une image, le logiciel a besoin d'un document rédigé en POV-Ray nécessitant au minimum trois éléments :

- une caméra qui prend 3 paramètres : la position de la caméra (coordonnées x,y,z) et l'endroit où la caméra regarde(coordonnées x,y,z)
- une source lumineuse qui prend deux paramètres : la source de lumière (coordonnées x,y,z) et la couleur de la lumière)
- un objet de type plane générant un sol

Nous pouvons ensuite générer différentes formes. Pour nos applications, nous avons choisis de traités les objets suivant :

- box : correspond à tout les parallélépipèdes, pour les créés, il faut

au minimum rentrer les coordonnées de deux points correspondant à deux points opposés du parallélépipèdes
- sphère : il faut au minimum indiqué les coordonnées x,y,z du centre de la sphère ainsi que son rayon pour pouvoir la créée
- cône : il faut indiquer les coordonnées et le rayon du cercle qui est à la base du cône et du cercle qui sera au sommet du cône, pour avoir un cône pointu, ce rayon devra être égale à zéro
- cylindre : il faut indiquer les coordonnées du centre des cercles qui sont à la base et au sommet du cylindre et un rayon qui sera identique pour les deux cercles

Il y a ensuite tout les paramètre optionnel qui sont communs à l'ensemble des formes que nous souhaitons créé :
- pigment : gère la couleur de la forme en prenant en argument un nom de couleur (nécessite d'implémenter une librairie contenant des constantes représentant des couleurs) ou son code RGB
- texture : gère la texture de la forme en prenant en argument le nom de la texture à appliquer sur l'objet (nécessite d'implémenter une librairie contenant un ensemble de texture)
- rotate : gère la rotation d'un objet par rapport à un ou plusieurs axe en prenant en argument un triplet d'entier correspondant à la rotation en degré par rapport à chacun des axes
- scale : modifie la taille de l'objet par rapport à un ou plusieurs axe.
- translate : modifie la position de l'objet par rapport à un ou plusieurs axe.

1.2 Les règles mathématiques

1.2.1 appliquées à la lumière

Les règles mathématiques que nous avons utilisé dans ce projet sont, pour la plus part, des règles de trigonométrie.

Afin de recréer au mieux l'action de la lumière sur un objet, nous avons suivit deux grands principes d'optique. Selon les lois de Snell-Descartes, lorsqu'un rayon lumineux entre en contact avec un objet, dans le cas d'une surface réfléchissante comme un parquet ciré ou un miroir, le rayon lumineux est réfléchit. Si l'objet n'est pas opaque et donc, que la lumière peut le tra-

versé, le rayon lumineux est réfracté.

Le rayon de lumière réfléchit par un objet se calcule en fonction de l'angle d'incidence du rayon qui est entré en contact avec une surface. L'angle d'incidence est l'angle entre le rayon lumineux et une droite perpendiculaire à la surface de l'objet passant par le point d'impact lumineux(on la nommera droite normale). Selon Snell-Descartes, cet angle est égal à celui entre la droite normale et le rayon réfléchit.

La réfraction d'un rayon lumineux se base sur les indices de réfractions des milieu traversé par la lumière.lorsque la lumière entre en contact avec un nouveau milieu, son rayon change de direction, il est réfracté. Selon Snell-Descartes, l'angle entre la droite normal et le rayon réfracté est égal à un arc cosinus de, l'indice de réfraction du milieu d'origine multiplié par le sinus de l'angle d'incidence, divisé par l'indice de réfraction du second milieu.

Afin de recréer ces principe en langage java, nous avons eu recourt à quelques fonction mathématiques :

- java.lang.Math.sqrt(double a) : Cette méthode prend en paramètre un double, lui applique la fonction racine carré et retourne soit un double, soit, si le paramètre entré est négatif, NaN.
- java.lang.Math.acos(double a) : Prend comme argument un double sur le quel on effectue un arc cosinus. Cette méthode nous retourne un angle en radian ou si la valeur absolue du double est superieur à 1 NaN.
- java.lang.Math.toDegrees(double a) : On entre en paramètre un angle en radian sous forme de double dont on veux récuperer la valeur sous forme de degrès. La valeur renvoyée est approximative.
- Droite.produitScalair(Droite a) : Prend en entrée une droite et calcule un vecteur pour chacune de ces droites puis, renvois le produit scalair de ces vecteur. On obtient un produit scalair de vecteur à trois entré en appliquant : vecteurA*vecteurB = X*X' + Y*Y' + Z*Z'.
- Droite.vecteur() : Cette méthode permet de renvoyer un vecteur d'une droite. Ce dernier se récupère en prennant deux point de la droite et dans le cas d'un plan en 3 dimensions, en faisant : vecteur = (X - X', Y - Y', Z - Z').

Ces lois de Snell-Descartes sont invariable dans le temps car, d'après le

principe de la relativité de Galilée, les lois physiques sont invariable et s'applique toujours quelque soit l'endroit où nous nous trouvons.

1.2.2 appliquées aux les formes

Pour la création des différents point d'incidence et droite normale de chaque objet, nous avons eu recourt à différentes équations paramétriques et équations cartésiennes. La droite normal est une droite perpendiculaire à un surface.

- Pour le cone : Afin de trouver la droite normal, il a fallut utiliser l'équation paramétrique du cercle servant de base au cone afin de récupérer le point commun entre la droite reliant le point d'incidence et le sommet et la base du cone. Pour le calcul du point d'incidence, nous avons utilisé l'équation cartésienne d'un cone et chercher la valeur du paramètre t, de l'équation paramétrique de la droite reliant le centre du cone et le point lumineux, en faisant un système à quatre équation.
- Pour le parallélépipède : La création de la droite normal c'est fait en calculant le vecteur orthogonal à deux autres vecteurs de la face sur la quelle la lumière tapait. Afin de trouver le point d'incidence nous avons utilisé le théorème de Möller Trumbore. Ce théorème s'appliquant au triangle, nous avons divisé les faces en deux triangles. Par la suite nous avons vérifié si il existé un point d'impact entre la droite reliant le point lumineux au centre du parallélépipède et les faces de ce dernier. Nous avons ainsi pu récupérer tout les points d'impact et ensuite, nous avons gardé celui dont la distance le séparant du point lumineux était la plus faible.
- Pour le cylindre : Pour retrouver la droite normal, nous avons procédé de la même façon que pour le cone à ceci près que nous avons dû calculer deux points d'intersection de cercle et de droite.C'en fût de même pour la recherche du point d'incidence pour la quelle nous avons usé de l'équation cartésienne du cylindre.

2 Organisation du projet

2.1 Répartition des taches

Nous avons réparti les taches par thématique, une partie de traduction du povray de manière à pouvoir l'utiliser dans le code par la suite, faite par Paul Lebranchu, une partie avec la création des différentes formes, faites par Marguerite Bauchez aidé par Olivier Cocquerez, une partie plus mathématique faite par Raphaëlle Lemaire.

Voici un tableau récapitulatif détaillant le rôle de chaque membre :

Nom :	Classes faites :	Méthodes hors Classes faites :	Classes Test faites :	autre :
Marguerite Bauchez :	Sphere, Point	Parallepipede : Constructeur, getPointDebut, getPointFin, sommet. Cylindre : Constructeur, getBas, getHaut, getRayon. Cone : Constructeurs, getBas, getHaut, getRayonB, getRayonH. Droite : Constructeurs, getPrem, getSec.		
Olivier Cocquerez	Sphere, Point	Parallepipede : Constructeur, getPointDebut, getPointFin, sommet. Cylindre : Constructeur, getBas, getHaut, getRayon. Cone : Constructeurs, getBas, getHaut, getRayonB, getRayonH. Droite : Constructeurs, getPrem, getSec.		Compilateur pour Raphaëlle

Nom :	Classes faites :	Méthodes hors Classes faites :	Classes Test faites :	autre :
Paul Lebranchu :	Room, Interprete, Render	Accesseur couleur dans les classes formes et implémentation javafx	Classe tests présentent dans le dossier Main (Translate, TestBox, TestCylindre, TestCone, TestSphere)	Compilateur pour Raphaëlle, script, exe.jar, fichier Povray, compilateur javadoc
Raphaëlle Lemaire :	Cercle, Plan, Vecteur, Lumiere, Shadow.	Parallepipede : rayonIncidant, estDansMiFace, estDansFace, planIncid, pointIncid, pointIntersect, droiteNormal, calculdist. Cylindre : recupDroiteLum, pointIncidence, pointCommunAvecCyl, calculDiscriminantCylindreEtDroite, droiteNormal. Cone : pointIncidence, valTSystEquaParam, droiteNormal. Droite :vecteurUnitair, calculLongueur, vecteur, vectCoordoA1, approximativeDroit, trouveT, represPram, calculAngle.	Lumiere : TestLum, TestSha. Forme : TestCercle, TestCone, TestCylindre, TestDroite, TestPlan, TestVecteur.	

2.2 Gestion du projet

Afin de mener à bien le projet de rendu 3D par lancer de rayon, nous avons eu recourt à divers source d'information telles que, https://docs.oracle.com/javase/8/docs/api/java, https://www.superprof.fr/ressources/scolaire/physique-chimie/seconde/optique. Pour la réalisation du rapport, nous avons utilisé les cours sur LaTeX de notre première année et

deuxième année de licence ainsi que différent logiciel utilisant LaTeX comme TeXworkInterpréteur ou Texmaker.

Nous avons également mis en place un cles classes tests pour le momentertains nombre de script pour faciliter l'exécution des programmes (compile.sh qui compile les différents fichier java, exe.sh qui exécute le programme, exeTest.sh qui compile et affiche le résultats des classes tests et cleaner.sh qui supprime les fichiers inutiles. Nous pouvons également trouver une archive .jar exécutable dans le projet qui permet de lancer l'application.

Durant le confinement, nous avons continué à travailler sur ce projet en postant nos avancés sur SVN et en faisant des appels vocaux grâce à des moyens de communication comme discord ou messenger. Nous avons également effectuer un grand nombre de commit pour pouvoir tester les fichiers de nos camarades dont le compilateur java ne fonctionnait pas sur l'ordinateur personnel.

3 Architecture du projet

3.1 Arborescence du projet

Lorsque nous faisons un checkout depuis notre zone de dépot SVN nous obtenons l'architecture suivante depuis la racine :

- un répertoire expression contenant notre rapport, notre fichier soutenance (format pdf) ainsi que deux dossier : rapport et soutenance contenant le fichier .tex ainsi que les images généré dans ce document
- un répertoire image qui contient les images créé par le logiciel
- un répertoire javadoc qui contient la javadoc relative à notre code
- un répertoire projet contenant notre projet organisé de la façon suivante :
  - un répertoire build qui contiendra tout les fichiers .class lorsque l'on compile notre code

    - un répertoire formes contenant l'ensemble des fichiers java concernant la création d'objet (parallélépipèdes, sphère, cone et cylindre) ainsi que diverse notion géométrique (point, droite, plan, vecteur, cercle)
    - un répertoire lumière contenant l'ensemble des fichiers java relatifs à la lumière et l'ombre c'est à dire un fichier Lumière, un fichier Shadow et un fichier Room (fichier créant la pièce et les objets)
    - un répertoire main contenant un fichiers qui interprète les fichiers POV-Ray et un fichier Render qui joue le rôle d'exécutable dans notre projet
    - un répertoire POV contenant des fichiers .pov pour effectuer des tests/démonstrations sur notre projet
- un répertoire script qui contient un script de compilation, un script lançant les classes tests et un script supprimant les fichiers inutiles
- une archive .jar exécutable qui lance le programme
- un fichier exe.sh qui lance notre application
- un fichier README.txt qui explique comment faire marcher notre programme

3.2 Diagramme de classe/module

Concernant notre projet de rendu3D, nous avons choisis de répartir notre programme en 3 packages :

- un package formes qui contient les classes nécessaire à la création des différentes formes à représenter et les éléments géométrique nécessaire pour leur construction ou calcul d'ombre par rapport à la lumière.
- un package lumière qui gère les effets d'ombres et de lumière et qui contient les méthodes qui génèrent les images au format png.
- un package main qui contient l'interpréteur des données issus du fichier POV-Ray et un exécutable qui génère les images en se basant sur les informations issus du parser POV-Ray et qui fait appel au méthode des packages formes et lumière.

Voici un diagramme résumant l'organisation des packages et leur interactions :

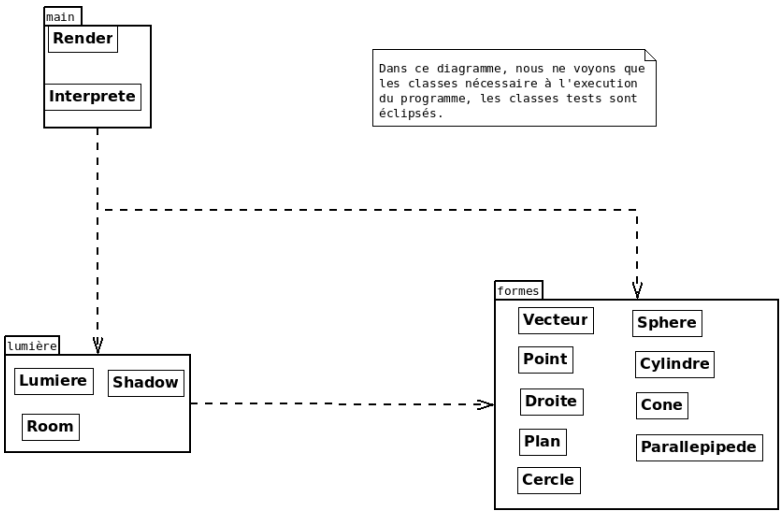


FIGURE 1 – Diagramme de classe générale

Sur ce diagramme, nous voyons que le package main est dépendant de ce que l'on trouve dans les packages lumiere et formes : pour pouvoir générer une image, la classe Render du package main devra faire appel au classe de formes pour générer les formes à mettre dans l'image et au packages lumiere pour pouvoir générer l'image à renvoyer grâce à la classe Room. On voit également que le package lumière est dépend du package formes : nous utilisons certains éléments créé dans le package formes dans les constructeurs et dans les méthode de ces classes (par exemple, la classe Shadow appelle la classe droite dans son constructeur et la classe Room appelle des points et des listes d'objets).

Le package main est organisé de la façon suivante :

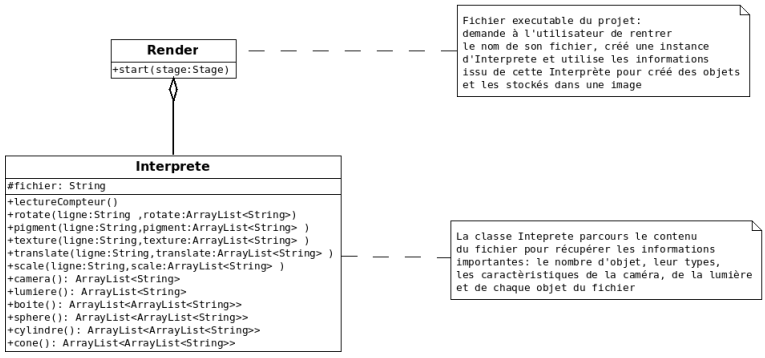


FIGURE 2 – Diagramme de classe du package main

Une classe Interprète se charge de récupéré les informations du ficher .pov en comptant le nombre d'objet présent dans le fichier et en récupérant leur caractéristiques (éléments nécessaires à la création et attribut optionnel) et en prenant les informations sur la caméra et la lumière et la classe Render demande à l'utilisateur de rentrez le nom de se fichier. Une fois qu'un nom de fichier valide a été inséré, la classe Render utilise les informations issus de l'instance d'Interprète de ce fichier pour créé différentes formes et met les informations dans une image au format png qui représentera la scène décrite par le POV-Ray.

Le package lumiere est organisé comme suit :

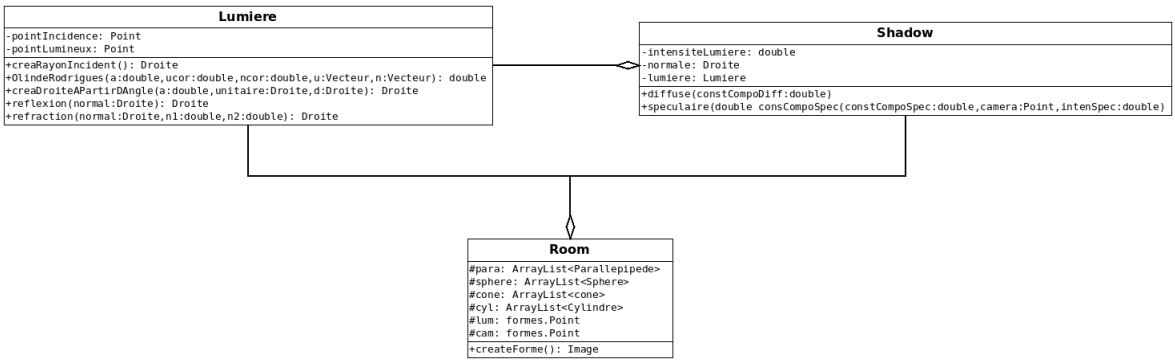


FIGURE 3 – Diagramme de classe du package lumiere

Le package lumière est constitué de trois classes. La classe Lumière qui crée les méthodes réflexion et réfraction, deux propriétés importante de la lumière l'une renvoyant un rayon lumineux comme le ferait un miroir tandis que l'autre renvois un rayon lumineux qui traverse l'objet (il faut que cette objet soit transparent pour que cette fonction puisse s'appliquer). Ces méthode sont réutilisée par la classe Shadow qui se charge de créer les ombres à la façon de phong et par Room qui crée l'image que le programme renverra sous format png grace à la classe Render du Main.

Le package formes est organisé ainsi :

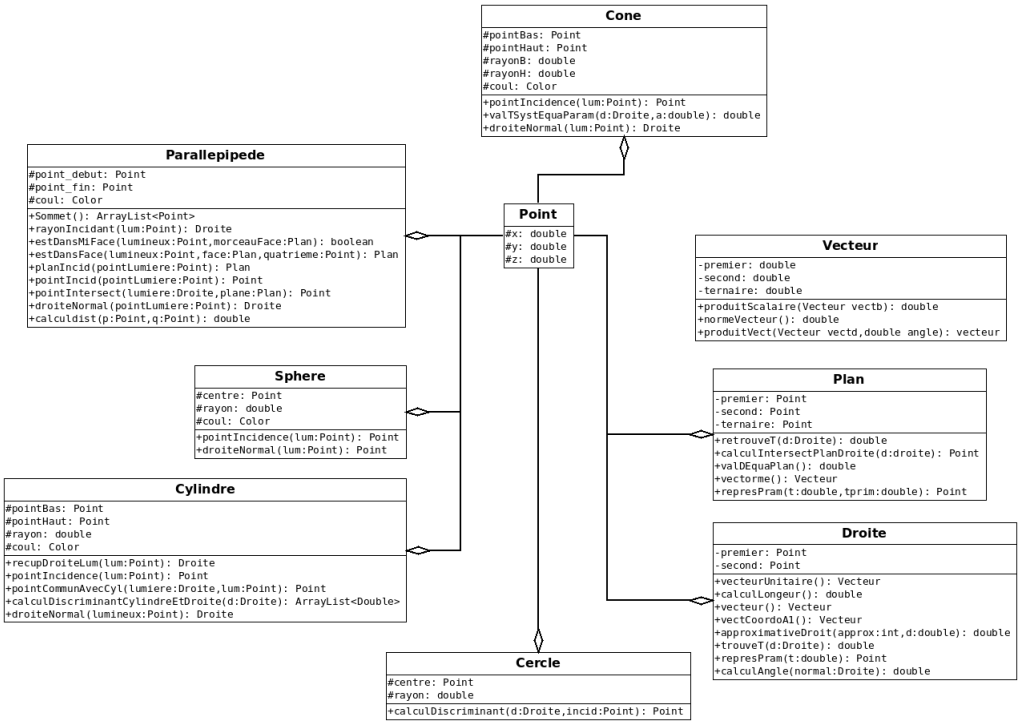


FIGURE 4 – Diagramme de classe du package formes

Dans ce package, toute les classes font appel à la classe Point, sauf la classe Vecteur. La classe Point permet la création des points pour chaque

figure mais elle sert aussi de base pour créer les droites et plans. Chaque figure à une méthode permettant de trouvé le point d'incidence de la lumière sur l'objet ainsi qu'une méthode pour retrouver la droite normal à la surface de l'objet au point d'incidence. Certaine classe comme Parallepipede on besoin de plusieurs méthode pour la création du point d'incidence.

4 Éléments Technique

4.1 Interprete et Render

La classe interprète a pour but de traiter les informations trouvait dans un document .pov et de renvoyer les informations essentielles du dit document. Dans le constructeur de cette classe, nous trouvons le fichier sous la forme d'un Objet String.

Nous allons maintenant voir la liste des différentes méthodes appelés dans cette classe :

- la méthode lectureCompteur parcours le fichier et compte les occurrences de chaque objets en repérant un mot clé (box, cylinder,sphere ou cone),stocke le résultat dans des compteurs et affiche la valeur de ces compteurs dans le terminal.
- viennent ensuite les méthodes récupérant les paramètres optionnels(rotate, texture, pigment, scale et translate). Ces méthodes ont été définis dans le but d'éviter la redondance dans les méthodes récupérant les informations sur les différents objets. Ces méthodes parcours le fichiers à la recherche de mots clé et une fois le mot clé trouvé, elle ajoute les informations à une liste. Par exemple, la méthode pigment recherche le mot clé pigment dans les lignes du document et si elle le trouve, elle ajoute le contenu qui se trouve après le mot clé dans un tableau placer en paramètre :

Algorithm 1: RÉCUPÉRÉ LA COULEUR D'UN OBJET

```
function Pigment (ligne, listePigment)
if ligne commence par "pigment" then
    if ligne contiens { et } then
        ligne ← ce qui est contenu entre { et }
        ajout ligne à listePigment
```

— les méthodes camera() et lumiere()renvoi toutes les deux des listes contenant des informations pertinentes vis à vis de la lumière et de la caméra (position,couleur de la lumière), contrairement aux objets (que nous verrons juste après), nous renvoyons une simple liste car il n'y aura qu'une seule instance de ces objets. La méthode camera cherche le mot clé caméra puis renvoie les informations concernant la position et l'endroit où regarde la camera. la méthode lumiere renvoi des informations concernant le point d'où est issu la lumière et la couleur de la lumière.

— les quatre dernières méthodes de la classe récupère les informations concernant les objets, elles parcours le fichier texte à la recherche des mots clés suivants : box (pour les parallélépipèdes), sphere, cone et cylindre. Une fois qu'elles ont trouvé ce mot clé, elle créé une liste qui contiendra les informations nécessaire à la création de l'objet (exemple : deux point pour une box, un point et un rayon pour une sphère), le paramètre concernant le rotation de l'objet puis sa couleur, sa texture, sa taille et son décalage par rapport aux axes x,y,z. Si un objet n'a pas de valeur pour un attribut optionnel, on ajoutera "none" à l'index de la liste où l'on était supposé trouvé ce paramètre (sauf pour le paramètre couleur où l'on rajoutera la couleur gris par défaut). Une fois que toutes les informations sur l'objet sont obtenus, on ajoute cet objet à une liste d'objets. Ces méthodes renvoie donc une liste constitué d'objets et ces objets sont décrits par une liste contenant les paramètres de l'objet.

la classe Render gère le rendu final de l'image par l'intermédiaire d'une application , elle demande à l'utilisateur de rentrer le nom d'un fichier dans le terminal, nous testons ensuite le fichier pour vérifier si il existe et si il

contient des objets que l'application pourrait afficher. Tant que l'utilisateur n'aura pas fourni de fichiers existant et disposant d'objet à afficher, le programme redemandera à l'utilisateur de rentrer le nom d'un fichier.

Une fois que le fichier aura été analyser et jugé comme valide par le programme, on créé les différents objets en parcourant les listes d'objets créé par les méthodes évoqués dans la classe Interprete ainsi que des points correspondant à la lumière et la caméra.

Lorsque tout les objets sont créés et stockés dans des listes d'objets, une instance de la classe Room dessine les formes dans une image et cette image est ensuite stocké dans un fichier. L'utilisateur doit ensuite rentrer le nom qu'il souhaite donné à son fichier et si le nom n'est pas déjà pris, alors le fichier est créé et contient une image, l'image est affiché sur l'écran (prévisualisation) et le programme se termine lorsque l'on ferme la fenêtre de prévisualisation de l'image (sinon, on demande à l'utilisateur de rentrer un nouveau nom pour son fichier).

4.2 Les méthodes et classes nécessaires à la création des formes

La classe Point permet la création de toute les figures du document et ainsi de connaître leur coordonnées dans l'espace. Elle prend en paramètre trois double et en fait un point au quel on peut changer chaque coordonnées et aussi les récupérer à l'aide des méthode getX ou setY.

La classe Droite permet de rentrer deux point en paramètre et, quand elle est appelé donne accès à diverse méthode :

- Les méthodes de création de vecteur, tel que vecteurUnitair permettant de créer un vecteur dont la norme vaut un. Ou encore vecteur qui permet de créer un vecteur en prenant les deux points de la droite. Ces méthodes permettent que, même en aillant une droite, si l'on a besoin d'un vecteur il suffit d'une seul ligne de commande.
- La méthode calculLongueur nous renvois la distance entre les deux points entrés en paramètre de la droite. Ce qui peut nous permettre de créer un nouveau point par exemple.
- La méthode represPram permet en sachant la valeur de t de récupérer un point, ce point est un point d'intersection entre notre droites et un autre objets mathématiques. La valeur de t est retrouvable en appli-

quant la valeur de x y z de l'équation paramétrique d'une droite dans l'équation cartésienne de l'objet mathématique du quel on cherche un point d'intersection.
- la méthode trouveT permet de calculer la valeur du paramètre t quand on cherche le point d'intersection de notre droite avec une autre.
- la méthode calculAngle renvois l'angle entre deux droites. En appliquant le calcul du cosinus d'un angle en trigonométrie. Ce dernier consiste à faire $cos(u, v) = u.v + ||u||x||v||$. Ainsi, pour récupérer la valeur de l'angle on fait un argcosinus du produit scalaire des vecteurs des droites plus la norme du premier vecteur fois la norme du second.

La classe Plan permet d'appliquer quelque fonction mathématique propre au plan, et ainsi de récupérer des valeurs primordiales aux calculs sur les faces. Cette classe implémente les méthodes suivante :
- La méthode retrouveT permet de calculer la valeur du paramètre t de la représentation paramétrique de la droite lorsque l'on cherche un point commun entre le plan et une droite.
- La méthode calculIntersectPlanDroite récupère la valeur de t et récupère le point calculé par la représentation paramétrique de la droite.
- La méthode valDEquaPlan qui calcul la valeur du d dans l'équation cartésienne du plan ($ax + by + cz + d = 0$(abc les vecteur normal et xyz les points recherché)).
- la méthode vectorme qui renvois un vecteur orthogonal à deux autre vecteur du plan. Ce vecteur est aussi appelé vecteur normal au plan et il est nécessaire au calcul de l'équation cartésienne
- la méthode represPram nous permet de trouver un point commun entre le plan et un objet mathématique à condition de récupéré au par avant un t et un t' paramètre de la représentation.

Les méthode de la classe Vecteur était en premier lieu calculer dans la classe droite et vecteur était instancié comme un point. Il a été nécessaire d'en faire une classe à par entière. Afin de pouvoir user de :
- La méthode produitScalair, cette méthode renvois la produit scalaire de deux vecteurs en appliquant $u.v = x * x' + y * y' + z * z'$. Avec xyz les coordonnées de u et x'y'z' les coordonnées de v.
- La méthode normeVecteur calcul la norme d'un vecteur, c'est à dire, sa longueur et direction. La norme d'un vecteur se calcul ainsi : $\sqrt{(}x + y + z)$.

— La méthode produitVect calcul le vecteur produit de deux vecteur, afin de faire marcher la méthode, il faut avoir récupéré la valeur de l'angle entre les deux vecteurs. Ainsi, on peux appliquer le calcul : $sinus(angle) * x * x'$ en remplaçant le x et x' par y et y' pour la coordonnée en y du point et par z et z' pour la coordonnée en z.

La classe cercle à été créer afin d'éviter la répétition et la dépendance des classes cônes et cylindre entre elle. Elle est composé d'une seule méthode calculDiscriminant permettant de récupérer le point d'intersection entre une droite et un cercle. Si l'on à deux point d'intersection, on récupère celui dont la distance le séparant du point lumineux et la plus faible.

4.3 Création des formes

A l'heure actuel, nous avons la possibilité de créer quatre formes primaires, un parallélépipède rectangle, un cone, une sphere et un cylindre. Chacune de ces formes à une méthode permettant de retrouver le point d'incidence de la lumière sur l'objet ainsi qu'une droite normal.

La création d'un parallélépipède rectangle dépend de la classe Parallepipede. elle nécessite deux points et une couleur. Les méthodes de cette classe sont :

— La méthode Sommet() qui à partir des deux points, calcule les six points restant du parallélépipède et renvois le tout dans une liste.
— La méthode rayonIncidant qui renvois la droite reliant le point lumineux au centre de l'objet. Pour cela, elle récupère le point central du parallélépipède en cherchant l'intersection des deux diagonales internes de la forme et crée la droite prenant pour coordonnée le point lumineux et le centre de l'objet.
— La méthode estDansMiFace se tir de l'algorithme de Möller Trumbore réajuster pour satisfaire notre code. On récupère un booléen qui nous dis si le point que nous cherchons est dans le triangle ou non. Le triangle dans notre code est la moitié d'une face.

Algorithm 2: Le point est t'il dans le triangle ?

function estDansMiFace ($pointLumineux$, $triangleABC$)
$epsilon \leftarrow 0.0000001$
$vecteurincident \leftarrow rayonIncidant(lumineux)$
$vecteurAB \leftarrow triangleB - triangleA$
$vecteurAC \leftarrow triangleC - triangleA$
$angle \leftarrow vecteurincident^{v}ecteurAC$
$vecteurH \leftarrow sinus(angle) * vecteurincident * vecteurAC$
$a \leftarrow vecteurAB * vecteurH$
if *a supérieur à -espilon et a inferieur à espilon* **then**
    retrouner faux
$f \leftarrow 1/a$
$vecteurLumA \leftarrow triangleA - pointLumineux$
$u \leftarrow f * vecteurH * vecteurLumA$
if *u supérieur à 1 et u inferieur 0* **then**
    retrouner faux
$angle2 \leftarrow vecteurLumA^{v}ecteurAB$
$vecteurQ \leftarrow sinus(angle2) * vecteurLumA * vecteurAB$
$v \leftarrow f * vecteurincident * vecteurQ$
if *$u + v$ supérieur à 1 et v inferieur 0* **then**
    retrouner faux
$t \leftarrow f * vecteurAC * vecteurQ$
if *t supérieur à epsilon* **then**
    retrouner vrai
retourner faux

— La méthode estDansFace renvois la partie de la face sous forme de plan contenant un point d'intersection avec la droite lumineuse . Elle vérifie ainsi le booléen renvoyé par l'algorithme de Möller Trumbore et renvois le plan qui a renvoyé vrai.
— La méthode planIncid vérifie la présence d'un point d'incidence pour chaque face de la forme et quand il y en a plusieurs, on prend le point le moins éloigné du point lumineux.
— La méthode planIncid vérifie la présence d'un point d'incidence pour chaque face de la forme et quand il y en a plusieurs, on prend le point le moins éloigné du point lumineux.Elle renvois le plan contenant le point recherché. Elle se complète avec la méthode pointIncid qui permet de

récupérer le point .
- La méthode pointIntersect renvois le point d'intersection entre une face de la forme et une droite.
- La méthode droiteNormal récupère les vecteurs directeur de la face et les utilise pour trouver grâce a la méthode vectorme le vecteur normal à la face.
- La méthode calculdist permet en rentrant deux point de récupérer la distance qui les sépares l'un de l'autre.

Pour créer un cylindre, on fait appel à la classe éponime.Le cylindre y est définit par deux points (les centres des cercles supérieur et inférieur) un rayon et une couleur. Afin de récupèrer la droite normale et le point d'incidence de la lumière sur l'objet, nous utilisons :
- La méthode recupDroiteLum qui créer la droite entre le point lumineux et le centre du cylindre.
- La méthode pointIncidence qui couplé avec pointCommunAvecCyl et calculDiscriminantCylindreEtDroite permet de récupérer le point d'incidence en calculant à l'aide de l'équation cartésienne du cylindre et de l'équation paramétrique d'une droite le paramètre t puis le point commun entre la droite et la forme. On cherche ensuite à renvoyer le point le plus proche de la source lumineuse.
- La méthode droiteNormal récupère le point du cercle bas et haut par le quel passe la droite basée sur le point d'incidence. Elle crée deux vecteurs un allant de incidence vert point bas et un autre de incidence vers point haut et utilise la méthode vectorme créant ainsi la droite normale.

Un cone se crée en entrant deux point, un pour le haut et un pour le bas, deux rayons pour les cercles du haut et du bas et une couleur. Tout comme les autres formes, on a créer de méthode pour récupèrer la droite normal et le point d'incidence.
- La méthode pointIncidence combinée à valTSystEquaParam récupère le point d'incidence en cherchant la valeur de t grace à l'équation cartésienne du cone puis en remplaçant t par la valeur trouvé dans l'équation paramétrique de la droite source lumineuse, centre de l'objet.
- La méthode droiteNormal cette méthode s'applique de façon similaire à celle du cylindre à une exeption près, au lieu de rechercher deux

points commun avec une droite et deux cercles, on en cherche un l'autre étant le sommet du cone.

Afin de créer une spère, on à besoin d'un point central d'un rayon et d'une couleur. La récupèration de la droite normal et du point incidence est ici plus simple que pour les autres formes car, nous avont déjà le centre de l'objet. Ainsi, nous utilisons :

- La méthode pointIncidence qui va chercher le point commun entre la sphere et le rayon lumineux.
- La méthode droiteNormal renvois la droite reliant le centre du cercle avec le point lumineux.

4.4 Gestion de la lumière

Pour pouvoir créer les ombres sur le objet, il était nécessaire de créer une classe Lumiere qui contient les méthodes permattant d'utiliser les fonction d'obtique lumineuse. De plus, il a fallut créer une classe permettant la création des ombres sur les objets grace à l'ombrage de phong. Pour créer un semblant de lumière sur des objets, nous avons créer la classe Lumiere contenant les méthodes :

- La méthode creaRayonIncident renvoyant une droite créer à partir de la source lumineuse et du point d'incidence de la lumière sur l'objet.
- La méthode OlindeRodrigues qui adapte la formule d'olinde-rodrigues permettant de récupèrer les coordonnées d'un vecteur une a une. Le vecteur récupèrer est le vecteur rentré en paramètre après application d'une rotation de x radian. La formule d'olinde-rodrigues est la suivante :

Algorithm 3: FORMULE D'OLINDE RODRIGUES

function Pigment ($vecteurSansRotation$, $vecteurUnitaire$, $angle$)
$vecteurObtenu = cosinus(angle) * vecteurUnitaire$
$vecteurObtenu+ = (1 - cosinus(angle)) * (vecteurUnitaire * vecteurSansRotation) * vecteurSansRotation$
$vecteurObtenu+ = sinus(angle) * (vecteurUnitaire * vecteurSansRotation)$
$retourner$vecteurObtenu

— La méthode réflexion renvois le rayon lumineux réfléchit par un objet. L'angle de ce rayon avec la droite normal à la surface est d'après DesCartes égale à celui entre le rayon incident et la droite normal. Cette méthode s'applique à tout objet réflechissant la lumière comme un miroir par exemple. On calcul donc l'angle entre la droite incidence et la droite normal et on applique la fonction d'olinde rodrigue pour récupèrer la droite réflechit.
— La méthode refraction s'applique à tout objet transparant. On applique à l'objet la formule de la réfraction, $n1 * sinus(angle1) = n2 * sinus(angle2)$ où n1 et n2 sont les indices de réfraction du milieu et les angle1 et deux les angles entre la normal et le rayon lumineux, permettant de récupèrer le rayon refracté par la surface.

La création des ombres se fait de part la méthode de phongen trois partie, deux d'entre elle sont recréer dans la classe Shadow :

— La méthode diffuse permet de recréer un effet d'ombre sur tout l'objet. Pour cela, on applique la formule de composante diffuse de phong :

Algorithm 4: FORMULE DE COMPOSANTE DIFFUSE DE PHONG

function Pigment ($constCompoDiff$)
$constCompoDiff \leftarrow valeurentre0et1$
$I = intensiteLumineuse * constCompoDiff * cosinus(angle)$

Dans cette formule, intensiteLumineuse est une valeur de l'intensitée lumineuse de la source et angle est la valeur de l'angle entre la droite normal et le rayon lumineux de plus, la constante de la composante diffuse dépendant de l'aspect de l'objet, plus l'objet et doux plus la valeur est proche de un.

— La méthode speculaire permet la création d"un point lumineux sur l'objet, il se crée garce à :

Algorithm 5: FORMULE DE COMPOSANTE SPÉCULAIRE DE PHONG

function Pigment ($constCompoSpec$)
$constCompoSpec \leftarrow valeurentre0et1$
$I = intensiteLumineuse * constCompoSpec * cosinus(angle)^{intensitSpeculaire}$

Dans cette formule, intensiteLumineuse est une valeur de l'intensitée lumineuse de la source et angle est la valeur de l'angle entre la caméra et le rayon réflechit de plus, la constante de la composante spéculaire dépendant de l'aspect de l'objet et, l'intensité spéculaire, plus elle est élevé, plus le point et petit et blanc.

4.5 Création de l'image

La création des formes ce fait à partir de la classe Room. Dans un premier temps, nous avions décider d'utilisé java swing pour représenter les formes mais nous nous sommes rendu compte assez tardivement que le résultat est loin d'être satisfaisant forme mal dessiné, position hasardeuse, défaut de perspective, impossibilité d'implémenter de quoi produire des effets d'ombre et de lumière).

Nous avons donc décidé de reprendre la modélisation dans un dossier annexe que nous avions nommé branche où nous retentions de modéliser les formes à l'aide de javafx. Une fois que le résultat obtenu dans la branche devenu de meilleur qualité que celle du programme en swing, nous avons supprimé la branche et gardé la version javafx de notre programme.

La classe Room prend en argument un ensemble d'ArraylList contenant les formes à afficher ainsi que deux points représentant la caméra (qui aurait du servir à gérer la caméra mais l'implémentation n'a pas réussi) et la lumière.

Cette Classe ne comporte qu'une seule et unique méthode qui créé l'image à dessiner : la méthode createForme (ne prend pas d'argument et renvoie une Image).

Dans un premier temps, Nous créons le fond de l'image. Pour cela, nous créons un point représentant la lumière et créons un effet Ligthning (effet lumineux), nous créons ensuite un rectangle gris foncé qui symbolisera le fond de l'image et nous appliquons l'effet lumineux dessus. Une fois cela fait, nous ajoutons ce fond dans un Group (élément contenant un ensemble de formes) qui sera stocké dans une Scene (permet la représentation du Group).

Nous représentons ensuite l'ensemble des formes en parcourant les listes mise dans le constructeur et créons les formes adaptés :

- Parallélépipède : Dans un premier temps, nous calculons la longueur, la hauteur et la profondeur de la boite. Puis nous créons une effet d'ombre portée, pour cela, nous avons besoin de créé un objet en 2D (DropShadow est un effet de javafx qui ne s'appliquent qu'au forme 2D). Nous créons donc un rectangle représentant la face avant de l'objet et lui appliquons l'effet DropShadow en faisant en sorte que si l'objet se trouve à gauche d'une source lumineuse, son ombre soit projeté à gauche (on fait la même chose si l'objet est à droite, en dessous ou au dessus de la source lumineuse). Nous ajoutons cette forme et cette effet au Group qui contient déjà le fond de l'image. Nous créons ensuite un objet de type Box qui prend en constructeur la longueur, la hauteur et la profondeur du parallélépipède et nous appliquons ensuite les méthodes translate pour le mettre à sa position et nous lui mettons une couleur grâce à l'effet setDiffuseColor puis nous l'ajoutons Group.
- Sphère : Pour créé une sphère, nous appliquons les même méthode que pour le parallélépipèdes mais en créant des formes différentes : nous créons dans un premier temps un objet de type Circle qui servira a représenter l'ombre de la sphère (demande un rayon en constructeur puis on effectue des translate pour positionner le cercle à la position de la sphère pour que la sphère puisse caché le cercle dessiné. Nous créons ensuite un objet de type sphère (la construction et le positionnement de la sphère marche de la même manière que pour un cercle) et nous lui ajoutons sa couleur avant de le mettre dans le Group.
- Cylindre : Lors de la création d'un cylindre,nous calculons dans un premier temps la hauteur du cylindre en comparant les coordonnées de l'axe Y du cercle du haut et du cercle du bas du cylindre. Nous créons ensuite un Polygon qui sera une représentation approximative de l'ombre du cylindre (il manque les arrondis) puis nous créons un

Cylindre (qui prend en paramètre un rayon et la hauteur du cylindre) avant de le mettre à sa place, de lui ajouter sa couleur et de l'ajouter au Group.

— Cône : Le dessin du cône est un cas particulier, contrairement aux parallélépipèdes, aux sphères et aux cylindres, il n'existe pas de classe prédéfini pour dessiner un cône. Nous avons donc décider de représenter les cônes d'une autre manière : dans un premier temps, nous créons des cercles représentant la partie base et la partie haute du cône ainsi qu'un polygone qui représente le corps du cône : ces formes servent à la création de l'ombre du cône. Nous dessinons ensuite des sphères au dessus des cercles puis un nouveau polygone par dessus celui nécessaire pour créé l'ombre du cône. Mais, un polygon étant une forme 2D, nous ne pouvions pas lui appliqué une couleur à l'aide de Phongmaterial. nous avons contourné le problème en replissant le polygone de la couleur du Cone et nous ui appliquons un effet de lumière pour donner l'impression que la forme est en 3D. Nous pouvons cependant constaté que les cônes ne sont correctement représentés uniquement lorsque les cercles du bas et les cercles du haut sont alignés par rapport à l'axe X.

Une fois la création de tout les objets terminé, nous prenons une photo de l'image généré par la scene et nous lui appliquons un effet de rotation (cet effet de rotation est créé pour que l'image en povray et l'image en javafx est la même origine (coordonnées (0,0,0). Nous créons ensuite un nouveau Group et une nouvelle Scene qui vont stocké l'image final. Puis nous créons un Stage qui affichera cette image : cela permet à l'utilisateur d'avoir un visuel de l'image qu'il vient de créé sans avoir à ouvrir de fichier. L'image est ensuite stocké dans une variable retourné par la fonction.

C'est dans cette classe que nous devions implémenter les effets des classes lumières et shadow mais nous n'avons pas réussi à implémenter ces classes dans la classe Room.

5 Manuel d'utilisation

5.1 Cas d'utilisation

Pour lancer notre application, il faut exécuter le scipt exe.sh dans un terminal ouvert à la racine du projet à l'aide de la commande suivante : "*./exe.sh*". Il se peut que vous ne disposiez pas des droits pour executer le script, dans ce cas, il faudra user de la commande "*chmod -x exe.sh*" sous windows ou "*chmod 774 exe.sh*" sous linux et relancer le programme à l'aide de la commande "*./exe.sh*".

Une fois le programme lancer, la console vous demandera de rentrer le nom du fichier POVRAY que vous dessiner, il faudra alors noter le nom d'un fichier POVRAY existant et qui se trouve dans le sous dossier pov du dossier projet (il faudra rentrer un chemin du type "*./pov/nomFichierPovray.pov*". Tant que vous n'aurait pas rentrer un nom de fichier valide, le programme vous redemandera de rentrer un nom de fichier.

Après avoir rentrer le nom de votre fichier, le programme affichera le nombre de forme et le type de forme que vous devrez trouverez dans votre image et vous demandera de choisir un nom de fichier. Si le nom du fichier est déjà pris, l'application vous montrera une image vide et vous demandera d'entrer un nouveau nom de fichier. Une fois que vous aurez rentré un nom valide, l'application vous affichera l'image créé et l'enregistra dans le dossier images qui se trouve à la racine du projet. L'application se termine dès que vous fermerez la fenêtre de prévisualisation de votre image.

Voici un exemple d'exécution et de prévisualisation de l'image avec le fichier exe.sh :

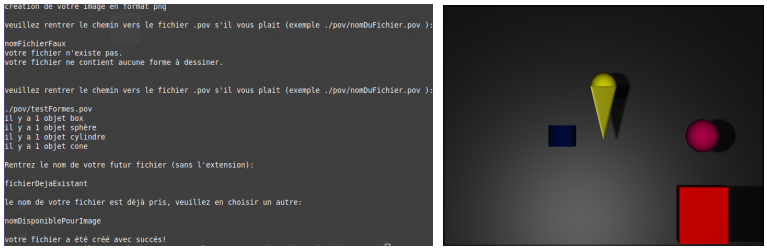


FIGURE 5 – Illustration de l'exécution en console avec exe.sh

Il est également possible de lancer notre programme depuis l'archive exe.jar à l'aide de la commande "*java -jar exe.jar*". Les conditions d'utilisations seront les même mais le chemin que l'on devra rentrer pour accéder au fichier POV-Ray sera différent (comme indiqué lors de l'execution du programme, nous ne devons plus rentré une adresse du type "*./pov/nomFichier.pov*" mais "*projet/pov/nomFichier.pov)*".

Voici un exemple utilisé sur la même image mais executé avec l'archive jar :

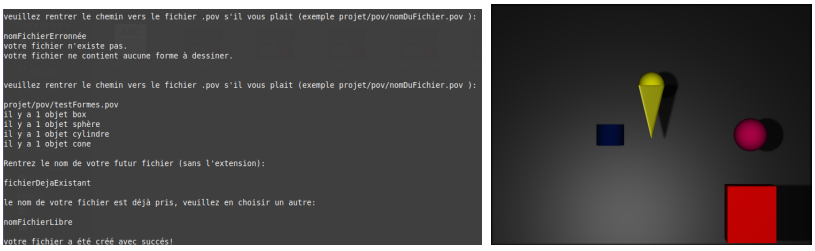


FIGURE 6 – Illustration de l'exécution en console avec exe.jar

5.2 usage

Nous allons maintenant afficher le résultat obtenu par notre application (à gauche) et le fichier obtenu avec l'utilisation de POV-Ray (à droite) et analyser les différences :

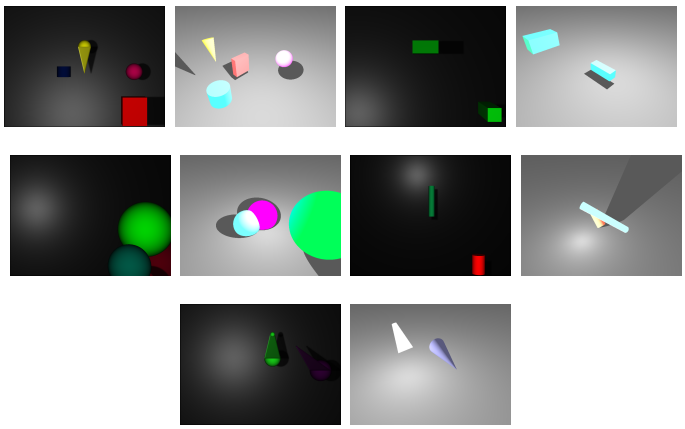


FIGURE 7 – comparatif fichier obtenu par l'application et fichier POV-Ray

Nous pouvons constater que sur chacune des images, les objets sont tous représentés et hormis les cônes, la représentations des formes sembles plutôt bien correspondent malgré quelque déformation, notament au niveau de la taille de certains objets.

Cependant, nous pouvons noté des erreurs qui sont du à la fois au placement (exemple sur l'image des sphères) mais également à la camera (comme sur l'image où toutes les formes sont représenter : si l'ont modifier l'emplacement de la caméra, nous pourrions retrouver une configuration quasi identique à celle du document POVRAY).

Le point lumineux ne semble pas être placé correctement et les effets d'ombre portée ne sont pas réalistes notament pour les cubes qui ne sont représenter que par un rectangle ou bien pour les cylindre donc les cotés arrondis ne sont pas représentés.

5.3 classes de test

Pour nous assurer du bon fonctionnent de notre programme et pour tester différents aspect de notre code, nous avons créé des classes de test. Ces classes ne sont pas destinés à être utilisées par le public mais elle devait nous servir à vérifier que nos codes marchaient correctement.

Chaque package disposé de ces classes de tests :

- Package formes :
  - TestCercle : Test de la méthode calcul discrimant.
  - TestCone : test des méthodes de création du cone et récupération de la droite normal et du point d'incidence.
  - TestCylindre : test des méthodes de création du cylindre et récupération de la droite normal et du point d'incidence.
  - TestDroite : test des méthodes création de vecteurs et récupération des point de croisement et angle entre les droites.
  - TestPlan : test des méthodes permettant de trouver les point d'incidence possible pour un plan/ face du parallélèpipède.
  - TestVecteur : test des méthodes permettant de réaliser les opérations de base des vecteur comme le produit scalaire, le produit vectoriel ou la récupération de la norme d'un vecteur.
- Package lumiere :
  - TestLum : test les méthodes permettant de créer les rayons reflechit, réfracté.
  - TestSha : test les méthodes permettant de récupèrer les composante nécessaire à l'ombrage de phong.
- Package main :
  - Translate : test la Classe Interprete sur un fichier POVRAY contenant différentes formes avec option ou sans option.
  - TestBox : vérifie que Interprète arrive à récupérer les informations essentiel à la création des parallélépipèdes (nombre de parallélépipèdes et informations nécessaires pour sa construction.
  - TestSphere : le même rôle que la classe TestBox mais pour les sphères.
  - TestCone : le même rôle que la classe TestBox mais pour les cônes.
  - TestCylindre : le même rôle que la classe TestBox mais pour les cylindres.

Conclusion

Les problèmes lié au confinement/matériel

Durant la phase de confinement, nous avons eu quelque problème de type matériel : certains de nos ordinateurs personnel n'était pas équipés pour compiler le code en java. Pour compenser cela, nous avons effectué de nombreux commit sur svn et les personnes qui disposaient de compilateurs java renvoyés les erreurs à la personne gérant le code. Nous avons également du faire face à des problèmes de connexions internet indépendant de notre volonté.

Les problèmes de codage et de mise en code

Dans la classe Lumière, lorsque l'on multiplie deux double entre eux, si ces deux double sont trop long, le résultat est Nand, pour palier à cela, il fut créé une méthode approximative qui renvois un double de la taille donnée en paramètre. Nous avons aussi eu un problème pour récupérer une droite à partir d'une autre droite et d'un angle, afin d'y remédier, nous avons fait appel à la méthode d'olinde Rodrigues qui permet d'effectuer une rotation d'un vecteur et ainsi de récupérer une droite.

Dans la classe pour la création du Parallèlépipede, nous avons réfléchit à une solution pour savoir si un point était contenu dans une face afin de récupérer le point d'incidence. Ainsi nous coupâmes chaque face en deux triangle afin d'appeler la méthode estDansMiFace qui implémente l'algorithme d'intersection de Möller Trumbore. Ainsi, si le point est contenu dans la face, nous cherchons si il est plus près du point lumineux que les autre points inclus dans d'autre face. Cela a permis de récupérer le point d'incidence du parallélépipède.

Pour les formes cylindre et cone, nous avons eu un problème pour trouver le point d'incidence sur un plan incurvé. Nous avons utilisé les équation paramétriques propre à chaque forme afin de retrouver leur équation cartésienne propre. Une fois l'équation récupérée, nous cherchions le paramètre t de l'équation paramétrique de la droite passant par le point lumineux et le centre de l'objet. Ainsi nous avons récupéré deux points puis il suffisait de choisir le plus proche de la source lumineuse.

Objectifs remplis

Nous avons réussi à créé un Interpréteur de fichier POV-Ray fonctionnel qui retourne les éléments principaux de ce fichier. Nous avons également réussi à généré un fichier au format png à partir de ce fichier et nous avons réussi à tracer des éléments dans ce fichier (bien que le résultats ne soit pas celui escompté). Nous avons également programmer de nombreuses méthodes effectuant des calculs pour géré le système d'ombre et de lumière mais nous n'avons pas réussi à l'implémenter dans notre fichier Room. Nous avons cependant réussis à créé un point lumineux ainsi que des effets d'ombre portée (qui n'a pas fini d'être développé) à l'aide des effets de javafx.

Piste d'amélioration

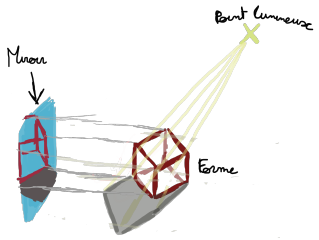
Pour améliorer notre projet, nous devrions revoir notre méthode d'affichage des objets : la position des objets, leur orientation (pour les cylindre) ainsi que la forme n'est pas toujours exact. De plus, il faudrait réussir à gérer les paramètres de la camera pour qu'il correspondent à ceux du povray.

Il serait aussi possible, de donner une transparence a un objet permettant à ce dernier d'être traversé par un rayon lumineux. Pour cela, aurait fallut mettre la couleur de fond de l'objet en un peu plus claire, et au endroit ou des faces de la formes donne une impression de superposition, foncé la couleur. La transparence de l'objet aurait permis l'utilisation de la méthode de réfraction de la lumière.

La création d'une méthode permettant la création d'ombre portée aurait permis de mieux visualiser l'objet dans l'espace. Cette méthode aurait été créée en, dans un premier temps cherchant si l'objet est en suspension ou poser. Si l'objet était en suspension, il aurait fallut trouver l'impact de la droite reliant chaque extrémité de la forme au sol et coloré l'intérieur de la projection de gris ou de la couleur du sol grisée. Si l'objet était posé, nous aurions récupéré le point d'intersection entre les extrémité de l'objet non inclut dans le sol et le point lumineux et, griser la projection résultante.

Nous aurions aussi pu créer des miroirs, renvoyant le lumière dans la pièce et ainsi multipliant les ombres des objets. De plus, ces miroirs aurait pu reflété la pièce. Afin de récupéré le reflet de la pièce, nous aurions procéder

en récupérant les différent point des figures inclus dans une forme, voir image ci-dessus.



Nous pourrions également ajouter une fonctionnalité permettant à l'utilisateur de choisir si il souhaite la conserver ou non l'image créé par l'application. Nous pourrions aussi faire en sorte que notre programme puisse prendre en argument une liste de fichier POV-Ray et qu'il génère une image pour chacun des fichiers valides se trouvant dans la liste.

Références

Java

— https://docs.oracle.com/javase/8/docs/api/index.html
— https://stackoverflow.com/
— https://ecampus.unicaen.fr/course/view.php?id=14884
— https://ecampus.unicaen.fr/course/view.php?id=14886

Mathématique/physique

— https://www.superprof.fr/ressources/scolaire/physique-chimie/seconde/optique

POV-Ray et son utilisation

— http://www.povray.org/
— http://www.f-lohmueller.de/pov_tut/basic/povtutf0.htm
— http://geomorph.sourceforge.net/povray/fr/Anatomie%20-%20fichier%20Povray.html
— http://www.f-lohmueller.de/pov_tut/x_sam/sam_360e.htm

Utilisation de javafx

— https://www.geeksforgeeks.org/javafx-box-with-examples/ (utilisé pour comprendre comment marcher une application javafx)