# 2 - Fichier de description de scène

## Objectif

Dans cette nouvelle étape, nous allons lire le fichier de description de scène.

## Spécification

### Format du fichier d'entrée

#### Commentaires

- Les lignes qui commencent par `#` sont des commentaires, elles doivent être ignorées.

#### `size` (obligatoire)

La premiere information est la taille de l'image qui doit être générée, largeur puis hauteur.

```
size 640 480
```

#### `output`

Ensuite vient le nom du fichier dans lequel l'image est générée
**Valeur par défaut** : `output.png`

```
output mascene.png
```

#### `camera` (obligatoire)

La position de la caméra.

```
camera x y z u v w m n o f
```

- `x y z` correspondent à la position de l'oeil (`lookFrom`)
- `u v w` correspondent au point visé par l'oeil (`lookAt`)
- `m n o` correspondent à la direction vers le haut de l'oeil (`up`)
- `f` correspond à l'angle de vue (field of view, `fov`) en degrés

**Exemple** : La caméra est positionnée en (0,0,4), regarde le centre de la scène (0,0,0),
la position haute suivant l'axe des y (0,1,0), avec un angle de vue de 45°

```
camera 0 0 4 0 0 0 0 1 0 45
```

## Couleurs

- `ambient` la couleur ambiante, en (r,g,b)
- `diffuse` la couleur de l'objet en (r,g,b)
- `specular` la lumière réfléchie (effet miroir) en (r,g,b)

**IMPORTANT** : il faut vérifier que (`ambiant` + `diffuse`) ne dépasse pas 1 sur chaque composante. Si c'est le cas, lancer une exception (entrée incorrecte).

```
ambient .1 .1 .1
diffuse 0.9 0 0
specular 0 0 0
```

Les couleurs `diffuse` et `specular` peuvent être déclarées à tout moment dans le fichier, à de multiples reprises. Dans ce cas, la dernière déclaration s'applique à l'objet défini ensuite.

```
diffuse .9 0 0
sphere 0 1 0 0.5

diffuse 0 0 .3
triangle 0 1 2
```

## Lumières

**`directional`**

`directional x y z r g b` indique une lumière globale directionnelle

- `x y z` représente une direction (vecteur)
- `r g b` une couleur

```
directional 0 0 1 .5 .5 .5
```

**`point`**

`point x y z r g b` indique une source de lumière locale

- d'origine un point (x,y,z)

- de couleur (r,g,b)

```
point 4 0 4 .5 .5 .5
```

Il peut y avoir plusieurs sources de lumières de chaque type, ou aucune.

**IMPORTANT** : il faut vérifier que la somme des couleurs des sources de
lumière ne dépasse pas 1 sur une des composantes. Lancer une exception sinon.

## Objets

Ensuite on va décrire les objets dans la scène
On dispose de trois objets de base :

- la sphère
- le triangle
- le plan

### Sphere (prioritaire pour ce projet)

`sphere x y z r` : on déclare une sphere à partir de son **centre** (x, y, z) et de son **rayon** r

```
sphere 0 1 0 0.5
```

### Triangle

Les triangles sont définis à partir de **trois points**.

Pour éviter d'avoir à mettre 3 coordonnées par définition de triangle,
on va définir en avance des points (appelés vertex ici), et les triangles seront ensuite définis par trois points.

### Vertexes / points

- `maxverts` : Il faut définir en avance le nombre de points avec `maxverts` (suivi d'un nombre entier)

```
maxverts 4
```

Ensuite chaque point est donné par ses coordonnées (pas forcément entières)

- `vertex x y z` pour un point de coordonnées (x,y,z)

```
vertex -1 -1 0
vertex +1 -1 0
vertex +1 +1 0
vertex -1 +1 0
```

#### Triangles

On peut maintenant définir des formes géométriques à partir des points.
Les points sont référencés par leur indice de déclaration, à partir de 0.
Toutes les formes sont créées à partir de triangles.

- `tri a b c` : un triangle de sommets d'indices (a, b, c) de la liste des vertexes

**Remarque** : on déclare deux triangles pour faire un carré

**Exemple** : un triangle de coordonnées (-1,-1,0), (+1,-1,0) et (+1,+1,0)

```
tri 0 1 2
tri 0 2 3
```

**IMPORTANT** : il faut vérifier que les indices sont strictement inférieurs à `maxverts`

#### Plans

`plane x y z u v w` : on déclare un plan à partir d'un **point** (x, y, z) de ce plan et d'un **vecteur** normal (u, v, w) à ce plan.

```
plane 0 -1 0 0 1 0
```

# Aide

## 1 - Pour valider

Vous trouverez des fichiers de scène valides dans le répertoire `scenes`.
Pour la suite (Jalon 3+), utilisez ces scènes, associées aux exemples de rendus pour valider le fonctionnement de votre projet. N'oubliez pas de vous servir du comparateur d'images pour vous aider à déterminer les différences.

## 2 - Le parsing

Deux possibilités :

- les expressions régulières
- la méthode `split` utilisée sur les espaces, et la méthode `startsWith`

Dans tous les cas :

- Faites un parcours ligne par ligne du fichier, traitez une ligne à la fois, dans l'ordre
- Nettoyez vos lignes avec `trim` pour éviter les espaces superflus
- Utilisez la structure `switch case` pour traiter les différents cas, et faites une méthode pour chaque cas

## 3 - Classes à créer pour le parsing

Il va falloir créer une classe `Scene` pour stocker les informations de la scène récupérées dans le fichier. Je ne saurais que trop vous conseiller de créer une classe dédiée au parsing du fichier `SceneFileParser`, qui va créer et modifier votre objet `scene` au fil du parcours du fichier. Ainsi, votre classe `Scene` contiendra juste des attributs, accesseurs et mutateurs (et d'éventuelles méthodes qui arriveront plus tard), mais ne sera pas encombrée de tout le code nécessaire à la lecture du fichier. Séparons les problèmes.

- N'oubliez pas de créer des conteneurs pour les formes et les lumières
- N'oubliez pas de convertir les données récupérées dans le bon type, car le fichier ne vous donnera que des `String`
  - `Integer.parseInt`
  - `Double.parseDouble`

## 4 - Classes à créer pour le reste

Vous aurez aussi à créer une classe par élément mentionné dans le fichier. Vous commencerez à avoir la structure du projet. Ajoutez des constructeurs à ces classes selon les attributs nécessaires.

- `Scene`, évidemment, qui contiendra :

```
private int width;
private int height;
private Camera camera;
private String output = "output.png";
private Color ambient = new Color();

private List<Light> lights = new ArrayList<>();
private List<Shape> shapes = new ArrayList<>();
```

- `AbstractLight` parent de `PointLight` et `directionalLight`
- `Shape` parent de `Sphere`, `Plane` et `Triangle`
- Il faudra stocker les couleurs `diffuse` et `specular` dans les objets concernés (les `Shape`). N'oubliez pas que cette information peut être définie plusieurs fois, alors récupérez la bonne !
- `Camera` avec les informations concernant la camera et l'oeil

À ce stade, le nombre de classes augmente, vous aurez probablement envie de faire un peu de rangement et créer des packages. Quelques idées :

- `geometry` : formes, points, vecteurs etc
- `raytracer` : tout ce qui va vous servir à tracer les rayons (lumières, scène et d'autres choses qui arriveront au jalon 3)
- `imaging` : les objets nécessaires à la génération d'images (Color pour le moment)
- `parsing` : ce qui sert à traiter votre fichier de configuration/description de scene (`SceneFileParser`)