UNIVERSITÉ DU LITTORAL CÔTE D'OPALE

# TP lancer de rayons 3
# Calculs d'intersections

Licence Informatique 3ème année

Année 2017-2018
durée : 3 heures

## 1 Introduction

L'objectif de ce troisème TP est de mettre en place les routines permettant de visualiser les objets présents dans la scène, en déterminant l'objet le plus proche de l'observateur en chaque pixel. Ce processus nécessitera de calculer l'intersection des rayons primaires avec les objets de la scène et de conserver l'objet le plus proche sur la trajectoire du rayon, s'il existe.

## 2 Représentation d'une intersection

Un point d'intersection ne peut être simplement représenté par un point, car il y a nécessité de conserver des informations plus larges que la simple localisation 3D de cette intersection. Il est en particulier nécessaire de savoir (i) sur quel objet se situe cette intersection et (ii) la distance (paramétrique) à laquelle elle se trouve de l'observateur. On propose alors la classe `Intersection` suivante (vous noterez qu'elle hérite de la classe `Point`, qui conservera les coordonnées 3D du point d'intersection) :

```
class Intersection : public Point {
private:
  Objet *objet;// objet sur lequel se situe l'intersection
  float distance;// distance paramétrique par rapport à l'origine du rayon
public:
  Intersection();
  Intersection(const Point& p, Objet* o, const float& t);
  ~Intersection(){}; // ne pas effacer objet !!!
};
```

Coder cette classe et ses premières fonctionnalités et la rajouter à votre application.

## 3 Calcul des intersections

Il s'agit à présent de calculer les intersections entre un rayon et les différents objets de la scène, pour, au final, ne conserver que la plus proche.

### 3.1 Calcul des intersections avec la scène

Dans un premier temps, vous allez mettre en place l'architecture générale des calculs d'intersection au sein des différents classes qui y font appel :

1. modifier la méthode `intersecte` de la classe `Scene` de telle sorte qu'elle corresponde au prototype suivant (rien à modifier dans le code de la classe à ce stade) :
   ```
   bool intersecte(const Rayon& r, Intersection& inter)
   ```
   avec :
   — `r` le rayon à intersecter avec la scène ;
   — `inter` la valeur de l'intersection trouvée, si elle existe.
2. faire de même avec les méthodes `intersecte` des classes `Objet`, `sphere` et `Plan`.

3. modifier la méthode `intersecte` de la classe `Scene` de telle sorte qu'elle effectue le calcul d'intersection du rayon avec tous les objets de la scène et retourne `true` **dès qu'elle trouve une intersection**, `false` sinon. Le paramètre `inter` contiendra la valeur de l'intersection trouvée si elle existe. Dans l'état actuel de votre code, aucune intersection ne sera trouvée.
4. modifier la méthode `genererImage` de la classe `Camera` pour effectuer le calcul d'intersection entre chaque rayon primaire et la scène. Si une intersection est trouvée, le pixel correspondant prendra la couleur bleu, sinon la couleur de fond. Il sera nécessaire d'ajouter une méthode permettant de récupérer la couleur de fond de la scène. Dans l'état actuel de votre code, aucune intersection ne sera trouvée.

## 3.2 Calcul d'une intersection avec une sphère

Il s'agit à présent de développer la méthode de calcul d'une intersection entre une sphère et un rayon. On rappelle :

— l'équation d'une sphère : $(x - xc)^2 + (y - yc)^2 + (z - zc)^2 = r^2$, avec $(xc, yc, zc)$ les coordonnées du centre de la sphère ;

— l'équation paramétrique d'un rayon : $R(t) = O + t.D$, avec $O = (x_O, y_O, z_O)$ les coordonnées de l'origine du rayon, $D = (x_d, y_d, z_d)$ sa direction et $t$ une valeur réelle quelconque. Les coordonnées de points situés sur un rayon vérifient alors :

$$\begin{cases} x(t) &= x_O + t.x_d \\ y(t) &= y_O + t.y_d \\ z(t) &= z_O + t.z_d \end{cases}$$

Il y a intersection entre un rayon et une sphère si le point correspondant verifie à la fois l'équation de la sphère et celle du rayon. En remplaçant les coordonnées d'un point $(x, y, z)$ dans l'équation d'une sphère par $(x(t), y(t), z(t))$, montrer que vous obtenez une équation du second degré en $t$ de la forme $a.t^2 + b.t + c = 0$, dont vous donnerez les coefficients.

— $a =$

— $b =$

— $c =$

À partir de l'expression de ces coefficients et de ce qui a été vu en cours sur la résolution de l'équation du second degré correspondante, compléter la fonction d'intersection de la classe `Sphere`, de telle sorte qu'elle calcule le point d'intersection entre le rayon et la sphère appelante, si il existe. L'application de cette nouvelle fonctionnalité sur les scènes `scene02.txt` et `scene03.txt` doit vous donner les images qui apparaissent sur les figures 1a et 1b.

(a) Résultat de l'application sur la scène `scene02.txt`

(b) Résultat de l'application sur la scène `scene03.txt`

FIGURE 1 – Aperçu des images obtenues à ce stade sur deux scènes de test.

## 3.3 Identifier l'intersection la plus proche

La version actuelle de la méthode `intersecte` de la classe `Scene` stoppe dès qu'une intersection est rencontrée. Ceci est incorrect, puisqu'il est nécessaire de garder l'intersection la plus proche, qui n'est pas forcément la première rencontrée. Pour vous rendre compte de l'erreur qui est commise, modifier votre application de telle sorte que la couleur affichée en cas d'intersection soit celle de l'objet trouvé, et non

(a) Résultat de la prise en compte de la couleur des objets sur la scène `scene02.txt`

(b) Résultat de la prise en compte de la couleur des objets sur la scène `scene03.txt`

FIGURE 2 – Aperçu des images obtenues en prenant en compte la couleur des objets sur deux scènes de test.

plus la couleur bleu par défaut. Vous devez obtenir les deux images qui figurent sur les figures 2a et 2b ci-après.

En examinant la figure 2b et le descriptif de la scène correspondante, vous noterez qu'elle est incorrecte, puisque la sphère verte est plus proche de l'observateur que la sphère rouge et qu'une petite sphère bleue située au premier plan n'apparaît pas. Pour obtenir l'image correcte de la scène `scene03` :

1. ajouter une méthode retournant la distance paramétrique t de l'intersection dans la classe `Intersection` ;
2. modifier la méthode `intersecte` de la classe `Scene` de telle sorte qu'elle examine toutes les intersections possibles et conserve la plus proche.

L'exécution de votre application sur la scène `scene03.txt` doit alors vous donner l'image correcte de la figure 3.

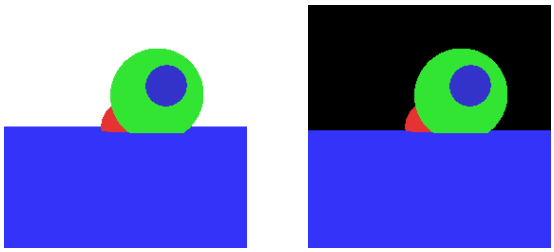
FIGURE 3 – Résultat de la prise en compte de la distance lors du calcul d'intersection sur la scène `scene03.txt`

## 3.4 Calcul d'une intersection avec un plan

On souhaite à présent pouvoir prendre en compte les plans dans les calculs d'intersection. On conserve la représentation paramétrique d'un rayon vue précédemment et on rappelle qu'un point de coordonnées $(x, y, z)$ appartient à un plan s'il vérifie l'équation :

$$a.x + b.y + c.z + d = 0$$

1. en utilisant le même type de raisonnement que pour la sphère, donner la formule permettant de déterminer la valeur du paramètre $t$ pour lequel existe une intersection entre un rayon et un plan ;
2. la valeur de $t$ est-elle toujours définie ? Dans la négative, expliquer dans quel cas elle n'est pas définie ;
3. quelles valeurs de $t$ sont valides ?
4. compléter la méthode d'intersection de la classe `Plan` et la tester sur le fichier `scene03.txt`. Vous devez obtenir le resultat apparaissant dans la figure 4a. En décommentant le plan noir vertical qui y figure, vous devez obtenir l'image de la figure 4b.



(a) Résultat de la prise en compte du plan horizontal bleu sur la scène `scene03.txt`

(b) Résultat de la prise en compte d'un second plan vertical oir sur la scène `scene03.txt`

FIGURE 4 – Aperçu des images obtenues en prenant en compte les plans figurant dans le fichier `scene03.txt`.