TP Initiation à la recherche 2

Master Informatique - 1ère année

Année 2020-2021

L'objectif de ce TP est d'étendre les fonctionnalités de l'application de lancer de rayons, à partir de documents issus de recherches sur les calculs d'intersection entre un rayon et une facette triangulaire. Cette primitive est effet celle qui est la plus utilisée en infographie, pour la modélisation d'objets complexes.

Exercice 1

Dans un premier temps, vous allez récupérer les fichiers `Triangle.hpp` et `Triangle.cpp` qui se trouvent dans l'archive fournie avec cet énoncé et les ajouter à votre application. Une instance de `Triangle` sera composée :

— d'un tableau de 3 `Point`, qui représentent chacun l'un des sommets du triangle.

— d'un vecteur, représentant la normale au plan du triangle (nécessaire pour les calculs d'éclairage).

Dans un second temps, il est nécessaire de modifier la lecture des fichiers de descrition d'une scène, de manière à pouvoir charger des triangles (méthode `charger()` de la classe `Scene`). La syntaxe proposée pour cette primitive géométrique sera la suivante :

```
tri x1 y1 z1 x2 y2 z2 x3 y3 z3
```

où (x_i, y_i, z_i) représente les coordonnées du sommet i. Deux nouvelles scènes vous sont fournies pour vos tests (`scene02.txt` et `scene03.txt`), à placer dans le dossier `Scenes` de votre dossier `LR`. On précise que seule la scène `scene02.txt` doit être utilisée à ce stade, la seconde scène étant réservée à des tests plus complexes par la suite.

Après avoir ajouté le code nécessaire à la lecture d'un triangle, compilez et testez votre application (évidemment, à ce stade, les triangles n'apparaîtront pas sur l'image ... ce serait trop simple :-)).

Exercice 2

Dans le constructeur paramétrique de la classe `Triangle` figure deux lignes de commentaires indiquant la manière dont la normale au triangle doit être calculée. Pour pouvoir compléter cette partie de code, vous devez ajouter une méthode de calcul du **produit vectoriel** dans la classe `Vecteur`(voir annexe). Lorsque cela sera fait, vous pourrez compléter ce constructeur avec les indications données.

Exercice 3

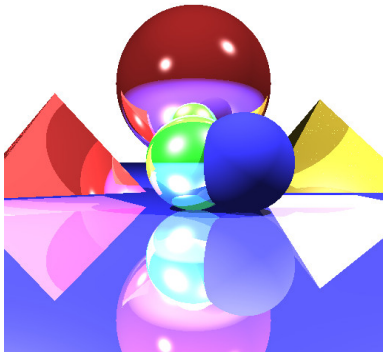
Vous allez à présent entrer dans le vif du sujet, en codant une première version des méthodes d'intersection pour la classe `Triangle`. Pour ce faire, récupérez le **document01** fourni, qui correspond à un support de cours de Brian Curless (Université de Washington [1]) et qui explique la technique basique pour calculer l'intersection entre un rayon et un triangle. Après avoir lu attentivement ce texte, implantez ce qu'il décrit dans la méthode `intersecte` de la classe `Triangle` et procédez aux essais. L'exécution de votre application sur le fichier `scene02.txt` doit vous donner l'image de la figure 1a.

Remarque : n'utilisez pas la classe `Plan` de l'application pour calculer l'intersection entre le rayon et le plan support du triangle, car l'intersection obtenue serait référencée dans le plan et non pas dans le triangle, ce qui poserait des problèmes lors des étapes suivantes de calcul ...

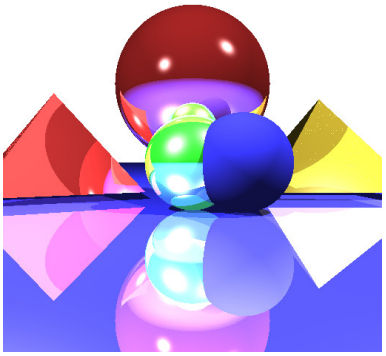
Exercice 4

Afin de pouvoir gérer les ombres produites par un triangle, il est nécessaire de compléter la méthode `coupe` de la classe `Triangle`. En s'appuyant sur la méthode décrite pour développer la méthode `intersecte`,

1. https ://courses.cs.washington.edu/courses/cse557/09au/lectures/extras/triangle_intersection.pdf



(a) Image obtenue après implantation de la méthode `intersecte` de la classe `Triangle`.



(b) Image obtenue après implantation de la méthode `coupe` de la classe `Triangle`.

FIGURE 1 – Effets des méthodes `intersecte` et `coupe` de la classe `Triangle` sur la scène `scene02.txt`.

peut-on se passer de calculer l'intersection avant de décider si un rayon coupe ou non le triangle ? En déduire le code de la méthode `coupe`. Après exécution de votre application sur le fichier `scene02.txt`, vous devez obtenir l'image de la figure 1b.

Exercice 5

De nombreux travaux de recherche se sont focalisés sur la manière la plus rapide de calculer l'intersection entre un rayon et un triangle. Parmi ceux-ci figurent les travaux de Tomas Möller et Ben Trumbore [2], qui ont proposé l'une des versions les plus utilisées à ce jour. L'article décrivant leur proposition est fourni avec cet énoncé, sous l'intitulé `document02`. Récupérez cet article et lisez le afin d'en comprendre le fonctionnement général.

Travail à réaliser

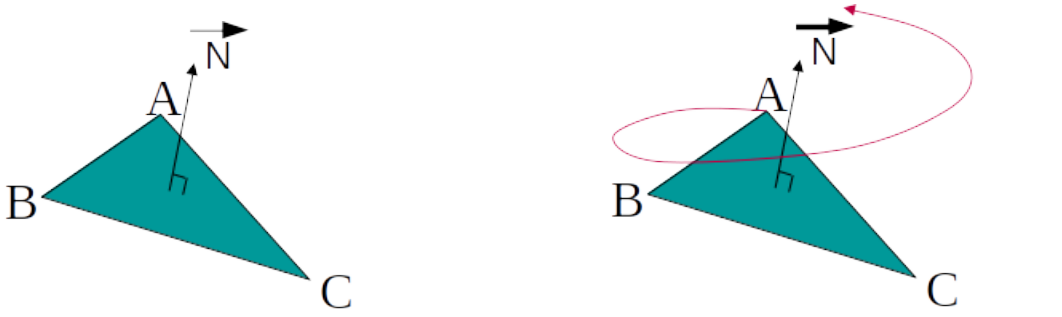
1. Afin de ne pas perdre la première version réalisée, vous allez placer le code des deux méthodes `intersecte` et `coupe` à l'intérieur d'une directive `#ifdef CURLESS .... #endif`. Ceci vous permettra ensuite, en définissant (ou non) la constante `CURLESS` d'activer (ou non) la compilation de ces méthodes ;
2. Implantez le code proposé dans l'article de Möller et Trumbore, en utilisant les fonctionnalités présentes dans l'application et non pas les macros utilisées dans l'article. Vous encadrerez le code des deux méthodes `intersecte` et `coupe` à l'intérieur d'une directive `#ifdef MOLLER .... #endif`, de la même manière que pour la version précédente. Testez le code pour vérifier que vous obtenez bien la même image que précédemment sur la scène `scene02.txt` ;
3. en compilant successivement chacune des deux versions (pensez à rajouter l'option -02 pour la compilation, afin de disposer d'une version rapide de votre application), évaluez les temps de calcul des deux approches sur la scène `scene03.txt`. A titre d'information, l'exécution sur un processeur I9 pour une image de taille 512×512 requiert une trentaine de secondes de calcul. Pour tester plus rapidement l'application, vous pouvez dans un premier temps réduire la taille de l'image à générer (par exemple 128×128). Seule une mesure approximative est demandée ; vous pouvez donc utiliser la commande unix `time` au lancement de l'application.

Annexe - Calcul de la normale à un triangle

Le vecteur normal, qui correspond à un vecteur perpendiculaire au plan du triangle (voir figure 2a), est très utilisé en synthèse d'images, car, d'une part, il fournit des informations sur l'orientation d'un triangle par rapport, par exemple, à l'observateur, et, d'autre part, il est nécessaire dans les calculs d'éclairage. Si l'on se refère à la figure 2a, on voit qu'un triangle possède deux normales : celle indiquée sur la figure et notée $\overrightarrow{N}$ et son opposée, le vecteur $-\overrightarrow{N}$.

Pour être cohérent dans les différents calculs impliquant la normale, il faut donc choisir un seul de ces vecteurs. La convention qui est adoptée est la suivante :

2. T. Möller, B. trumbore, Fast, Minimum Storgae Ray-Triangle Intersection, Journal of Graphics Tools, Vol. 2, No 1, PP 21-28, 1997, http ://dx.doi.org/10.1080/10867651.1997.10487468)



(a) Normale positive à un triangle ABC. (b) Sens de rotation du produit vectoriel $\overrightarrow{AB} \times \overrightarrow{AC}$.

FIGURE 2 – Illustration du mode de calcul de la normale à un triangle à partir du produit vectoriel $\overrightarrow{AB} \times \overrightarrow{AC}$.

Convention : La normale à un triangle doit toujours être orientée vers l'extérieur de l'objet que le triangle (et ses voisins) approxime.

Pour respecter cette règle, les logiciels de modélisation sauvegardent les sommets de chaque triangle dans un ordre bien précis, appelé sens direct, qui assure que la normale correcte pourra être calculée par le produit vectoriel suivant :

$$\overrightarrow{N} = \overrightarrow{AB} \times \overrightarrow{AC}$$

Ce produit vectoriel est illustré sur la figure 2b, dans laquelle on voit que le vecteur $\overrightarrow{AB}$ tourne autour de la normale vers le vecteur $\overrightarrow{AC}$ dans le sens trigonométrique, ou encore en suivant la règle très connue du *tire-bouchon* : on tourne dans le sens horaire pour enfoncer le tire-bouchon et dans le sens trigonométrique pour l'extraire ...

Dans l'exemple de la figure 2b, si les noms des sommets B et C étaient échangés, le formule du produit vectoriel ci-dessus donnerait un sens de rotation horaire et la normale serait dirigée vers le bas.