Curless versus Moller
pour l'intersection d'un triangle avec un rayon

Vitse Maxime - M1 I2L (Ingénierie du logiciel libre) à l'ULCO
Février 2022
maxime.vitse@decathlon.com

Abstract

Brian Curless a publié en avril 2004 une méthode d'intersection d'un rayon avec un triangle qui porte son nom. Huit ans plus tard, en avril 2012, Tomas Möller et Ben Trumbore ont eux aussi publié une autre méthode permettant l'intersection d'un rayon avec un triangle. J'ai implémenté ces deux algorithmes que je vais comparer dans cet article.

KEYWORDS : Ray tracing triangle, Curless, Möller, comparison

Introduction

Les deux algorithmes ont été implémentés dans un code source déjà existant avec un système de scènes, d'objets et de caméras permettant le lancer de rayons réalisé par Christophe Renaud [1]. Les algorithmes sont disponibles sur le repository github suivant : https://github.com/Weamix/InitiationALaRecherche. Dans la classe Triangle.cpp, vous pouvez les retrouver dans les méthodes intersecte et coupe. Pour activer Möller, il faut définir la constante "MOLLER" sinon, par défaut, c'est l'algorithme de Curless qui est utilisé.

Différences entre les deux algorithmes

Dans les deux algorithmes, on part d'un triangle avec trois sommets et on cherche à savoir si un point est à l'intérieur d'un triangle ou s' il est à l'extérieur pour savoir s'il y a une intersection. La différence réside seulement dans la méthode de calcul utilisée pour savoir si le point est dans ou en dehors du triangle. Les deux méthodes vont donc être détaillées dans les deux parties suivantes pour bien comprendre la différence.

Explication détaillée de Curless

L'algorithme de Curless est basé sur trois points A,B,C qui sont les trois sommets d'un triangle et le point Q qui est : origine du rayon + t * vecteur directeur du rayon avec t :
(d - normale à la surface du triangle * vecteur origine) / (normale à la surface du triangle * vecteur directeur du rayon)
Avec d qui est la normale à la surface du triangle multipliée par le vecteur du sommet 0 (A).

Il y a trois conditions, chacun des produits vectoriels multiplié par la normale à la surface du triangle qui sont vérifiés pour savoir si le point est en dehors du triangle, dans le cas où ils sont

égaux ou inférieurs à 0, entre les vecteurs :
- BA et QA
- CB et QB
- AC et QC

Sinon, il y a intersection car le point est bien à l'intérieur du triangle.

Explication détaillée de Möller

Tandis que l'algorithme de Möller tire son avantage de la paramétrisation de P, le point d'intersection en termes de coordonnées barycentriques. Les coordonnées barycentriques peuvent être utilisées pour exprimer la position de tout point situé sur le triangle à trois scalaires. L'emplacement de ce point comprend toute position à l'intérieur du triangle, toute position sur l'un des trois bords des triangles ou l'un des trois sommets du triangle eux-mêmes. Pour calculer la position de ce point à l'aide de coordonnées barycentriques, l'équation suivante est utilisée : P=uA+vB+wC

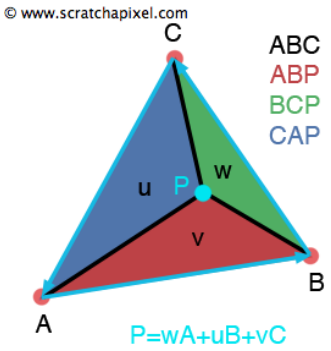

Figure 1. Coordonnées barycentriques

A, B et C sont les sommets du triangle et u,v,w trois nombres réels tel que u+v+w=1 alors le point barycentrique P est normalisé.

La valeur de w peut être trouvée avec celles de u et v car w = 1 - u - v donc u + v ≤ 1

Le point P est dans le triangle si et seulement si 0≤u,v,w≤1, sinon le point est en dehors du triangle.

L'implémentation de Möller est donc le calcul des nombres réels u et v.

u = (tvec * pvec) * inv_det;
Avec tvec qui est la distance depuis le sommet 0 (A) à l'origine du rayon,
pvec qui est le produit vectoriel du vecteur directeur du rayon avec le vecteur entre le sommet 0 et 2 (par exemple A-C)
Puis inv_det qui est l'inverse du déterminant.
Le déterminant étant le vecteur entre le sommet 0 (A) et le sommet 1 (B) multiplié par pvec.

v = (qvec * r.direction) * inv_det;
Avec qvec qui est le produit vectoriel de tvec avec le vecteur entre le sommet 0 (A) et le sommet 1 (B).

Une fois que nous avons u et v, nous appliquons des simples conditions pour vérifier si le point P est en dehors du triangle pour respecter la formule: $0 \leq u,v,w \leq 1$

donc si $u \,||\, v < 0$ ou si $u \,||\, u + v > 1$
Sinon il y a une intersection avec le triangle.

Performances

J'ai testé ces deux algorithmes sur une même machine dans les mêmes conditions de performances sur une dizaine de lancements pour m'assurer de la cohérence des résultats. Les deux algorithmes sont lancés sur deux scènes différentes en calculant le temps de rendu. Les scènes sont disponibles dans le dossier "scenes" à la racine du repository.

La première scène (scene02.txt) consiste au rendu de plusieurs sphères (4) et 2 pyramides visibles, ce qui fait au total 12 triangles en comptant les différentes faces et bases de triangles sur la scène.
La deuxième scène (scene03.txt) est un masque d'une tête de singe qui est une scène beaucoup plus complexe, composée de 968 triangles, elle est donc beaucoup plus compliquée.

Pour mieux visualiser, voici des rendus des 2 scènes :

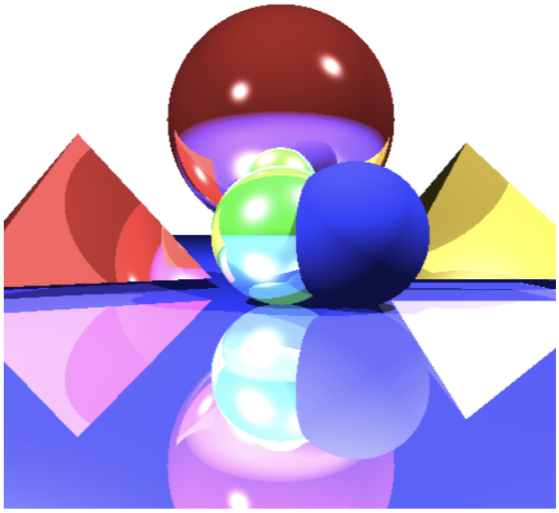

Figure 2. scene02.txt

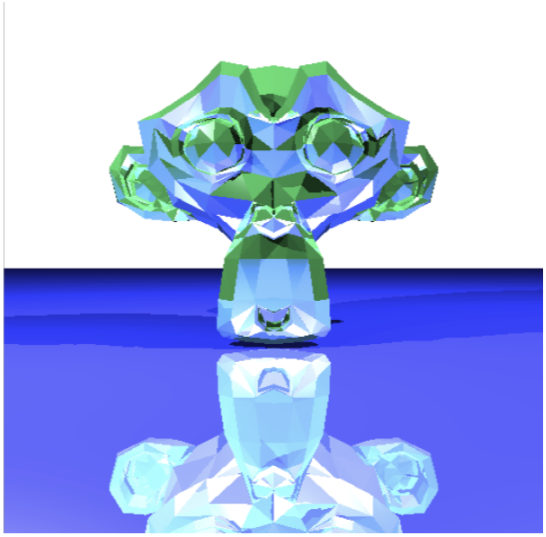

Figure 3. scene03.txt

A l'aide de la commande time d'unix, j'ai pu créer ce tableau récapitulatif des temps d'exécutions :

scene02.txt		
Unix time	Curless	Möller
real	4.131s	3.544s
user	2.029s	1.704s
sys	1.780s	1.793s

scene03.txt		
Unix time	Curless	Möller
real	1m36.438s	57.970s
user	1m34.305s	55.691s
sys	0m1.857s	2.189s

A titre indicatif, les tests ont été faits sur un ordinateur avec un processeur 2,3 GHz Intel Core I9 8 coeurs pour une image de taille 512*512 pixels.

Les mesures approximatives des deux algorithmes montrent sans aucun doute possible que l'algorithme Möller est le plus rapide par rapport à celui de Curless.

Conclusion et futures recherches

Le ray tracing coûte cher en puissance de calcul pour avoir le rendu. Chaque rayon a besoin d'avoir une intersection avec chaque objet de la scène. L'intersection d'un objet avec un rayon représente 95% du temps d'intersection selon Andrew Woo et John Amanatides [4].

Les chercheurs essaient donc de réduire le plus possible ce temps de rendu.

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 qui ont proposé l'une des versions les plus utilisées à ce jour. Les performances calculées ci-dessus entre les deux algorithmes le prouvent bien.

L'algorithme de Möller est universellement reconnu et utilisé.
Cependant dans un article publié en 2016, Doug Baldwin et Michael Weber [5] indiquent dans l'abstract que leur algorithme est toujours plus performant que celui de Möller :

"Running under ideal experimental conditions, our algorithm is always faster than the standard Möller and Trumbore algorithm, and faster than a highly tuned modern version of it except at very high ray-triangle hit rates. Replacing the Möller and Trumbore algorithm with ours in a complete ray tracer speeds up image generation by between 1 and 6%, depending on the image."

Références

[1] [Christophe Renaud - 2021/2022] https://www-lisic.univ-littoral.fr/~renaud/

[2] [Curless - 2004] Ray-triangle intersection

[3] [Moller & Trumbore - 2012] Tomas Möller & Ben Trumbore (1997): Fast, Minimum Storage Ray-Triangle Intersection, Journal of Graphics Tools, 2:1, 21-28

[4] [Andrew Woo et John Amanatides 1987] A Fast Voxel Traversal Algorithm for Ray Tracing

[5] [Doug Baldwin Michael Weber - 2016] Doug Baldwin and Michael Weber, Fast Ray-Triangle Intersections by Coordinate Transformation, *Journal of Computer Graphics Techniques (JCGT)*, vol. 5, no. 3, 39-49, 2016

[6] Scratchapixel - Ray Tracing: Rendering a Triangle