Path Tracer

Etudiant parcours M2 GICAO :

Aymeric Seguret

Henry Lefèvre

Enseignant : Romain Vergne

15/03/2016

Description de la scène utilisée

Par soucis de simplicité, la scène est décrite comme une fonction implicite. On peut ainsi utiliser la propriété de la distance signée dans l'algorithme du ray-casing pour trouver rapidement l'intersection entre un rayon lancé et un objet. La scène se compose d'un plan horizontal et de 3 sphères alignées.

C'est la fonction *distanceField* qui assure de renvoyer la distance cherchée à partir d'un point de l'espace. A noter qu'il y a un second paramètre retourner en plus de la distance qui est l'indice de l'objet qui a été touché. Cet indice permet par la suite de définir facilement un matériaux par objet (*cf. fonction getMateriau*).

Algorithme du path-tracer

Pour reprendre, l'idée est de faire rebondir aléatoirement un rayon lancé depuis la caméra jusqu'à frapper l'environnement (ou lorsque la profondeur max a été atteinte).

Dans un premier temps, l'ensemble des rayons qui forment le chemin parcouru par la lumière depuis la caméra vers l'environnement est empilé dans un tableau.

La couleur d'un rayon contribuant à son prédécesseur, l'algorithme dans un second temps part du dernier rayon et remonte jusqu'au premier lancé par la caméra en accumulant successivement toutes les couleurs trouvées grâce à la fonction *globalIllumination*.

Cette fonction, *globalIllumination*, prend entre autre en entrée la lumière incidente au point désirée et le vecteur vu. Elle retourne la couleur qui est perçue par l'observateur qui serait situé sur ce vecteur vu. A chaque nouvelle itération, le précédent vecteur vu devient le rayon de lumière incidente sur le rebond qui suit.

La figure ci-dessous illustre le dépilement successif des rayons accumulés jusqu'à l'environnement. Figure de gauche :

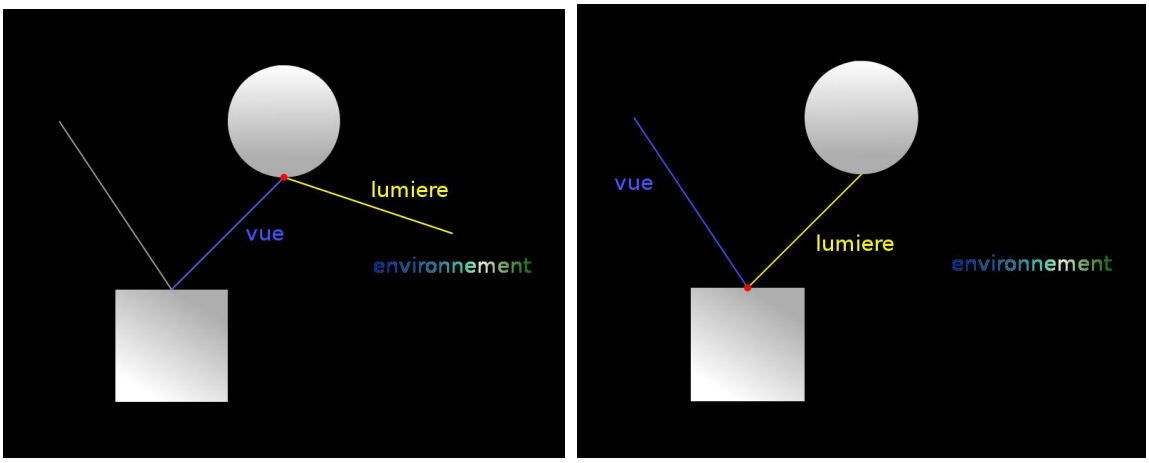
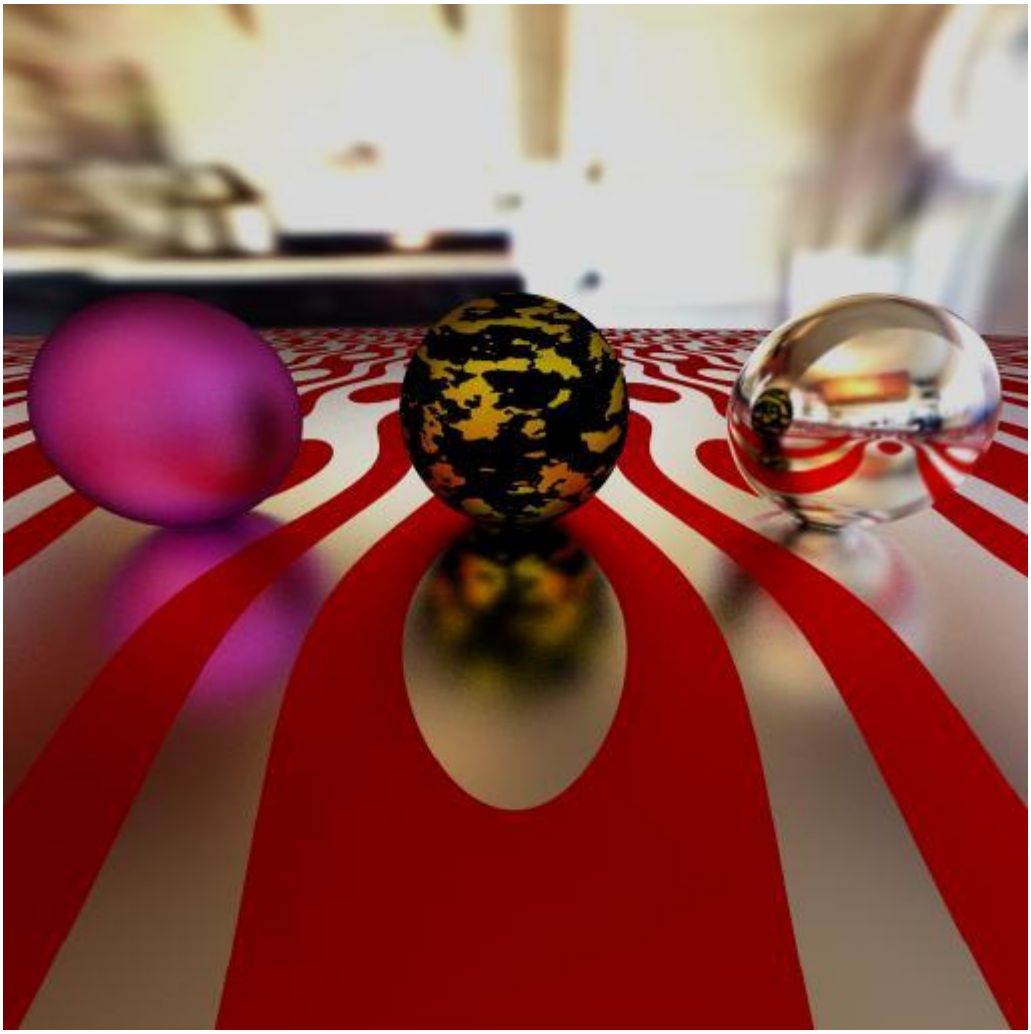


Figure gauche : le premier rayon dépilé contient la couleur de l'environnement et la couleur sur le cercle est calculé à partir du vecteur vu.

Figure droite : l'ancien vecteur vu devient le rayon lumineux incident. Et la couleur du cube est calculé à partir du nouveau vecteur vu dépilé.

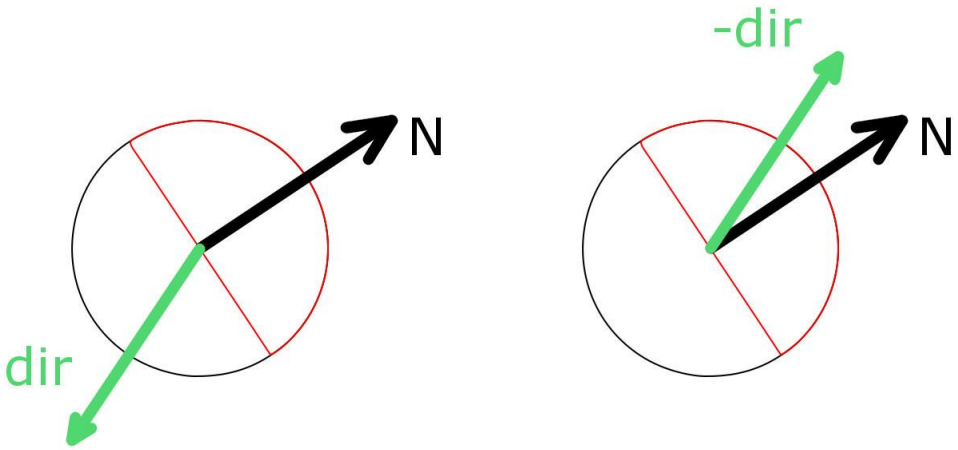
Le résultat final apparaît ci-dessous :



Méthode d'échantillonnage de la demie-sphère

La première idée pour sortir un vecteur aléatoire sur une demie-sphère (dont l'orientation dépend d'une normale donnée en entrée) reposait sur le principe suivant :

- Choix d'un vecteur aléatoire sur la sphère
- Si le produit scalaire entre le vecteur aléatoire et la normale est inférieur à 0 alors on prend le vecteur opposé



A première vue, cette méthode nous semblait correct mais force est de constater que les résultats en sortie du path-tracer n'étaient pas concluants. Les raisons à cela sont restées obscures à nos yeux. L'échantillonnage devait être de mauvaise qualité voire complétement faux.

Nous avons donc mis en place une autre solution pour résoudre ce problème. Cette fois-ci le principe est de choisir un vecteur aléatoire sur la demie-sphère supérieure. Puis de calculer son équivalence dans l'espace tangent TBN au point désiré.

Cette méthode donne des résultats plus cohérents que la première. Par exemple, la réflexion de la bille bleue sur la bille de droite n'apparaissait pas avec la première méthode.

L'anti-aliasing

Pour réduire l'effet d'aliasing présent dans le rendu final chaque pixel dans le FragmentShader est échantillonné 8 fois sur sa surface (*cf. variable pixelSampling*). Un échantillonnage uniforme est réalisé et la couleur finale du pixel est la moyenne de ses 8 rayons.

Remarque : on aurait aussi pu lancer un seul rayon aléatoire dans chaque pixel est faire x nombre de passe en plus. Le résultat est le même dans les deux cas.

L'importance sampling

La prise en compte d'une nouvelle méthode d'échantillonnage permet, à partir du paramètre de rugosité d'un matériau de réfléchir dans une direction plus ou moins proche du rayon réfléchi donné.

Le plan focal

Le flou de profondeur a été implémenté à partir des résultats précédents mais dans un nœud à part par soucis de lisibilité. Le principe est le même que pour le path tracer. Sauf que pour chaque pixel, la camera est placée quelque part aléatoirement sur le disque qui entoure la position initiale de la

caméra. Au lieu d'appeler une fois la fonction *pathTracing*, une étape préliminaire consiste à trouver le point de l'espace situé sur le plan focal lié au pixel. Ensuite seulement, la fonction pathTracing est appelée autant de fois que nécessaire mais avec une rayon lancé qui part de la lentille et passe par le "point focal". La moyenne est faite sur tous ces rayons lancés.