

ÉCOLE
CENTRALELYON

ÉCOLE CENTRALE LYON

MOS 2.2 : INFORMATIQUE GRAPHIQUE

TP – Introduction au Raytracing

Elèves :

M. Abderrahim OUESLATI

Responsable :

Pr. Nicolas BONNEEL

Table des matières

1	Introduction	2
2	Principe	3
3	Réalisation et résultats	4
3.1	La classe Vector	4
3.2	La classe Ray	4
3.3	La classe Object	4
3.4	La classe Sphère	4
3.5	La classe Triangle	5
3.6	La classe Scène	7
4	Éclairage et résultats	7
4.1	Éclairage direct	7
4.2	Ombre portée	8
4.3	Surface miroir	10
4.4	Surface transparente	11
4.5	Éclairage étendue	12
4.6	Éclairage indirect	12
4.7	Le maillage	15
5	Conclusion	17

1 Introduction

Dans ce BE, il y a une présentation de quelques techniques de "Raytracing". Le raytracing (« lancer de rayons » en français) est une technique de calcul d'optique par ordinateur, utilisée pour le rendu en synthèse d'image ou pour des études de systèmes optiques. Elle consiste à simuler le parcours inverse de la lumière : on calcule les éclairages de la caméra vers les objets puis vers les lumières en simulant le comportement et l'interaction lumière-matière, alors que dans la réalité la lumière va de la scène vers l'œil (caméra).

Cette technique reproduit les phénomènes physiques (principe du retour inverse de la lumière de Fermat, lois de Snell-Descartes...) que sont la réflexion et la réfraction. Une mise en œuvre naïve du ray tracing ne peut rendre compte de phénomènes optiques tels que les caustiques, l'illumination globale ou encore la dispersion lumineuse, pour celles il faut une approche plus élaborée du ray tracing, faisant appel à des techniques probabilistes de type méthode de Monte-Carlo, Metropolis2 ou à la radiosité pour résoudre ces problèmes.

En revanche, contrairement à d'autres algorithmes de synthèse d'image, elle permet de définir mathématiquement les objets à représenter et non pas seulement par une multitude de facettes. Les méthodes par lancer de rayon sont généralement lentes, mais simulent très bien et assez facilement toutes les interactions lumière-matière. Elles sont plus adaptées à l'industrie du cinéma ou de la simulation physique de l'éclairage, qui se permettent plusieurs heures de calcul par image.

Le but de ce TP établie sur plusieurs séances, est de se familiariser avec les principes fondamentaux du raytracing et faire implémenter un programme raytracer manipulant des primitives simples (sphères, plans, triangles). On améliorera ce raytracer en un “path-tracer”, tout en incluant les effets de l'éclairage indirect.

2 Principe

Cette technique de rendu consiste, pour chaque pixel de l'image à générer, à lancer un rayon depuis le point de vue (la « caméra ») dans la « scène 3D ». Le premier point d'impact du rayon sur un objet définit l'objet concerné par le pixel correspondant.

Des rayons sont ensuite lancés depuis le point d'impact en direction de chaque source de lumière pour déterminer sa luminosité pour savoir s'il est éclairé ou à l'ombre d'autres objets ?. Cette luminosité, combinée avec les propriétés de la surface de l'objet (sa couleur, sa rugosité, etc.), ainsi que d'autres informations éventuelles (angles entre la normale à l'objet et les sources de lumières, réflexions, transparence, etc.), déterminent la couleur finale du pixel.

On peut simuler aussi, plusieurs rebonds qu'un rayon lancé de la caméra peut faire avec les différents objets de la scène en faisant un appel récursives à la fonction qui calcule la couleur des pixels.

Cette technique fonctionne à rebours des modèles physiques qui, eux, lancent des rayons de lumière depuis les sources lumineuses vers l'œil ou la caméra en passant par les objets, alors que le ray tracing procède de la caméra vers les sources de lumières. L'expérience montre en effet que cette manière de procéder est nettement plus performante dans la plupart des cas.

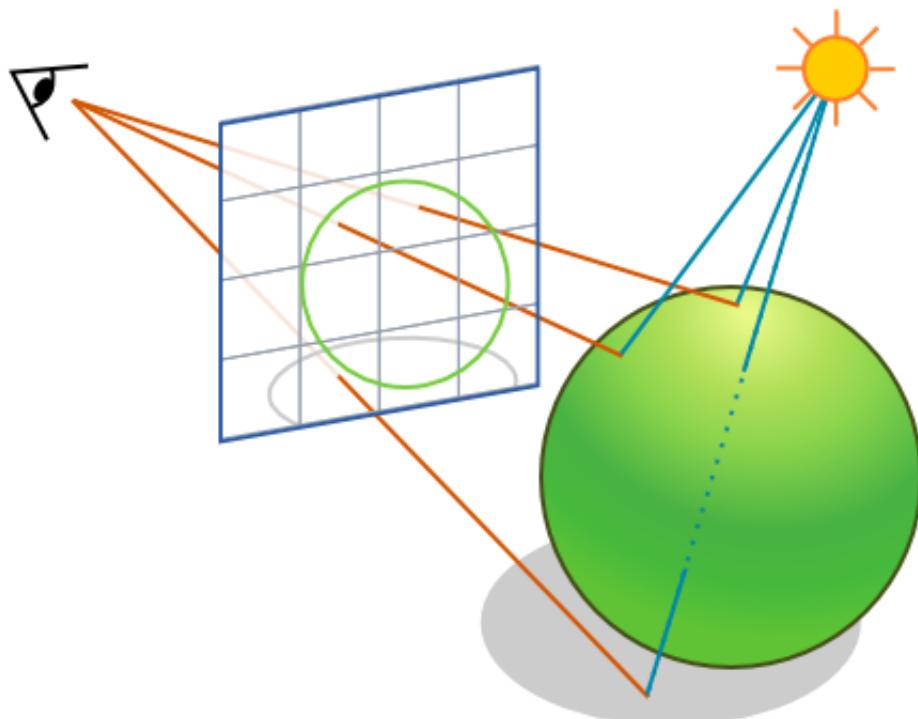


FIGURE 1 – Principe de "RayTracing"

Différentes techniques permettent d'accélérer ce traitement lourd. Dans le cadre d'objets complexes constitués de triangles par exemple (maillage), on peut dessiner l'objet de façon simple qui permet de déterminer précisément quels pixels seront réellement visibles, et permet de ne lancer de rayons que pour ces pixels limités.

Ces calculs préliminaires peuvent être réalisés par un processeur dédié (GPU par exemple) afin de limiter le traitement à faire par le processeur principal. Le raytracing étant très gourmand en calcul vectoriel. Donc il faut essayer d'optimiser au maximum les calculs.

3 Réalisation et résultats

On utilise tout au long de ce TP le langage de programmation C++. Pour commencer, on considérera une scène composée des sphères et/ou des triangles.

3.1 La classe Vector

On construit une classe Vector, fondamentale en 3D, puisque tous les composants des vecteurs sont en 3D et même les points dans l'espace, peut être représentés par cette classe. elle contiendra des coordonnées x, y et z en double précision. On fait aussi le surcharge des opérateurs : addition, soustraction de deux vecteurs, multiplication par un scalaire, le produit scalaire, la normalisation, la norme d'un vecteur au carré, le produit vectoriel.

3.2 La classe Ray

Cette classe Ray, représente un rayon de lumière : une origine et une direction unitaire (vecteur normalisé).

3.3 La classe Object

On définira une classe Object : c'est une classe générique qui contient les propriétés commune des objets qui peuvent être présents dans la scène et qui sont :

- l'albedo qui est un vecteur qui représente la couleur de l'objet et qui a trois composantes : rouge, vert, bleu. cet attribut est très important il sert beaucoup dans le calcul des couleurs finales à mettre dans chaque pixel de l'image finale.
- miroir ("mirror") c'est un attribut booléen qui indique si l'objet a une surface miroir et reflète la lumière (surface spéculaire) sinon une surface diffuse elle réfléchira la lumière dans toutes les directions, indépendamment de la direction incidente.
- transparent un attribut booléen qui indique si l'objet a une surface transparente (qui refracte la lumière) ou pas.
- une méthode abstraite "intersect()" qui va calculer le point d'intersection , s'il existe, d'un rayon donné avec la surface de l'objet. cette méthode doit être strictement implémenter par tous les classes filles qui héritent de la classe Object.

3.4 La classe Sphère

Cette classe hérite de la classe Object d'une manière publique. la classe sphère représente bien évidemment une sphère donc elle est composée par :

- une origine "O" : qui est une instance de la classe Vector mais elle représente le point origine de la sphère.
- un rayon "R" : c'est un double (réel) qui représente la valeur de rayon de la sphère.
- Puisque cette classe hérite de la classe object donc elle a son albedo et ses caractéristiques surfaciques(miroir, transparent) et bien sûr elle doit implémenter forcément sa propre méthode d'intersection "intersect()" qui calcule s'il y a une intersection de la sphère avec un rayon R(C,V) et calcule le point d'intersection "P" et la normale à la surface à ce point comme suit :

$$\|P - O\|^2 = R^2 \quad (1)$$

Or l'ensemble des points P décrits par le rayon de lumière s'écrit sous la forme :

$$P = C + t.V \quad (2)$$

où V est le vecteur directeur unitaire du rayon de lumière, C son origine, et t un paramètre le long du rayon.

Les points satisfaisant ces deux équations sont donc donnés par l'ensemble des t tels que :

$$\|C + t.V - O\|^2 = R^2 \quad (3)$$

et par suite :

$$t_2 + 2. \langle V, C - O \rangle t + \|C - O\|^2 - R^2 = 0 \quad (4)$$

Ce qui donne équation à un inconnu de degré deux, qui peut avoir 0 ou deux solutions (ou deux solutions dégénérées en une seule). En considérant toujours la solution t1 inférieure à la solution t2 qui est le point plus proche et visible par rapport à la caméra.

3.5 La classe Triangle

Cette classe hérite de la classe Object d'une manière publique. la classe Triangle représente bien évidemment un triangle dans l'espace 3D donc elle est composée par :

- Trois Vector A, B et C qui sont réellement des points représentants les trois sommets
- Puisque cette classe hérite de la classe object donc elle a son albedo et ses caractéristiques surfaciques(miroir, transparent) et bien sûr elle doit implémenter forcément sa propre méthode d'intersection "intersect()" qui calcule s'il y a une intersection de la sphère avec un rayon R(C,V) et calcule le point d'intersection "P" ses coordonnées barycentrique α , β et γ et la normale à la surface à ce point comme suit : Pour déterminer le point d'intersection entre un rayon et un triangle, On commence par trouver le point d'intersection "P" entre le rayon et le plan dans lequel appartient ce triangle. Un plan peut se définir par la donnée d'un point P_0 lui appartenant et par une normale N à celui-ci. L'équation de ce plan est alors donnée par l'ensemble des points P tels que le vecteur P_0P est perpendiculaire à N donc :

$$\langle \vec{P_0P}, N \rangle = 0 \quad (5)$$

De la même manière que pour la sphère, le point d'intersection rayon-plan est donnée par le point, qui satisfait à la fois la condition ci-dessus, ainsi que l'équation du rayon

$$P = C + t.V \quad (6)$$

On obtient donc :

$$\langle C + t.V - P_0, N \rangle = 0 \quad (7)$$

Enfin l'intersection :

$$t = -\langle C - P_0, N \rangle / \langle V, N \rangle \quad (8)$$

Après le calcul du point d'intersection avec le plan du triangle, il faut vérifier si ce point se situe bien à l'intérieur du triangle ou bien à l'extérieur. S'il est à l'extérieur, c'est que le rayon n'a finalement pas intersecté le triangle. On calcule alors les coordonnées barycentriques d'un point P en 2D sur le plan de triangle par rapport à ses 3 sommets A, B et C. On a alors :

$$\alpha + \beta + \gamma = 1 \quad (9)$$

$$\alpha \vec{PA} + \beta \vec{PB} + \gamma \vec{PC} = 0 \quad (10)$$

Si les coordonnées barycentriques sont toutes entre 0 et 1, alors le point d'intersection "P" se situe à l'intérieur du triangle. Résoudre ce système à 2 inconnues par la méthode de Cramer. En pratique, on a :

$$\vec{AP} = \beta \vec{AB} + \gamma \vec{AC} \quad (11)$$

et on projette sur les vecteurs et , on obtient les deux équations :

$$\vec{AP} \cdot \vec{AB} = \beta \|\vec{AB}\|^2 + \gamma \vec{AC} \cdot \vec{AB} \quad (12)$$

$$\vec{AP} \cdot \vec{AC} = \gamma \|\vec{AC}\|^2 + \beta \vec{AC} \cdot \vec{AB} \quad (13)$$

donc sous une forme matricielle :

$$\begin{bmatrix} \vec{AP} \cdot \vec{AC} \\ \vec{AP} \cdot \vec{AB} \end{bmatrix} = M \cdot \begin{bmatrix} \beta \\ \gamma \end{bmatrix} \text{ avec } M = \begin{bmatrix} \|\vec{AB}\|^2 & \vec{AB} \cdot \vec{AC} \\ \vec{AB} \cdot \vec{AC} & \|\vec{AC}\|^2 \end{bmatrix} \quad (14)$$

Enfin :

$$\beta = \frac{\det \begin{bmatrix} \vec{AP} \cdot \vec{AB} & \vec{AC} \cdot \vec{AB} \\ \vec{AP} \cdot \vec{AC} & \|\vec{AC}\|^2 \end{bmatrix}}{\det(M)} \quad (15)$$

et

$$\gamma = \frac{\det \begin{bmatrix} \|\vec{AB}\|^2 & \vec{AP} \cdot \vec{AB} \\ \vec{AB} \cdot \vec{AC} & \vec{AP} \cdot \vec{AC} \end{bmatrix}}{\det(M)} \quad (16)$$

3.6 La classe Scène

c'est une classe qui représente toute la scène 3D et qui va contenir l'ensemble des objets géométriques. Cette classe contient :

- objects : un tableau dynamique ("vector") de pointeurs sur la classe Object. Ces pointeurs pointent sur les différents objets géométriques présents dans la scène.
- un réel qui représente l'intensité lumineuse
- un pointeur sur sphère "L" qui représente la source lumineuse. En pratique, on a commencé d'abord par une source lumineuse ponctuelle (sous forme de classe Vector) puis, on passe à un éclairage étendu.
- méthode "intersect()" : elle parcours tous les objets de la scène (du tableau objects) pour appeler la fonction de calcul d'intersection de chaque objet de la scène avec un rayon donné et on prend le point le plus près et visible à partir du caméra.
- méthode "getColor()" qui calcule la couleur finale (rouge, vert, bleu) d'un pixel donné à partir d'un rayon et un nombre maximal de rebond.

4 Éclairage et résultats

Les opérations fondamentales dans un raytracer sont la génération de rayons, et le calcul d'intersections entre un rayon et un objet géométrique primitif.

Étant donnée la configuration de notre caméra centrée au point C, de champs visuel "fov" = 60 degrés, et regardant fixement vers les z négatifs. Pour générer un rayon vers le pixel (i, j), On génère une direction grâce au vecteur : $V = (j\text{-largeur}/2+0.5, i\text{-hauteur}/2+0.5, -hauteur/(2\tan(\text{fov}/2)))$ puis de le normaliser, on ajoute une origine, qui est le point C.

Une fois le rayon généré, on peut ensuite tester si ce rayon intersecte un objet géométrique en appelant tout simplement la fonction d'intersection des objets déjà implémentés.

4.1 Éclairage direct

On considère au début que le matériau de la sphère dans la scène est diffuse, l'intensité du pixel est calculé selon l'équation suivante :

$$pixel(i, j) = \max(0, \vec{l} \cdot \vec{n}) * I/d^2 \quad (17)$$

où \vec{l} est un vecteur unitaire qui part du point d'intersection P et se dirige vers la source de lumière L ponctuelle, \vec{n} est la normale de la sphère au point d'intersection, I est l'intensité de la lumière et d est la distance du point d'intersection à la lumière.

Dans un premier temps, on ne considérera pas les différents rebonds de la lumière à travers la scène, mais uniquement l'éclairage direct c'est à dire l'éclairage reçu par une source de lumière ponctuelle directement (pas de source étendue), et non l'éclairage ambiant dû aux autres interactions, et sans ombre portées. Dans notre scène on met juste une simple sphère et on ignore les murs et le sol, on obtient l'image suivante :

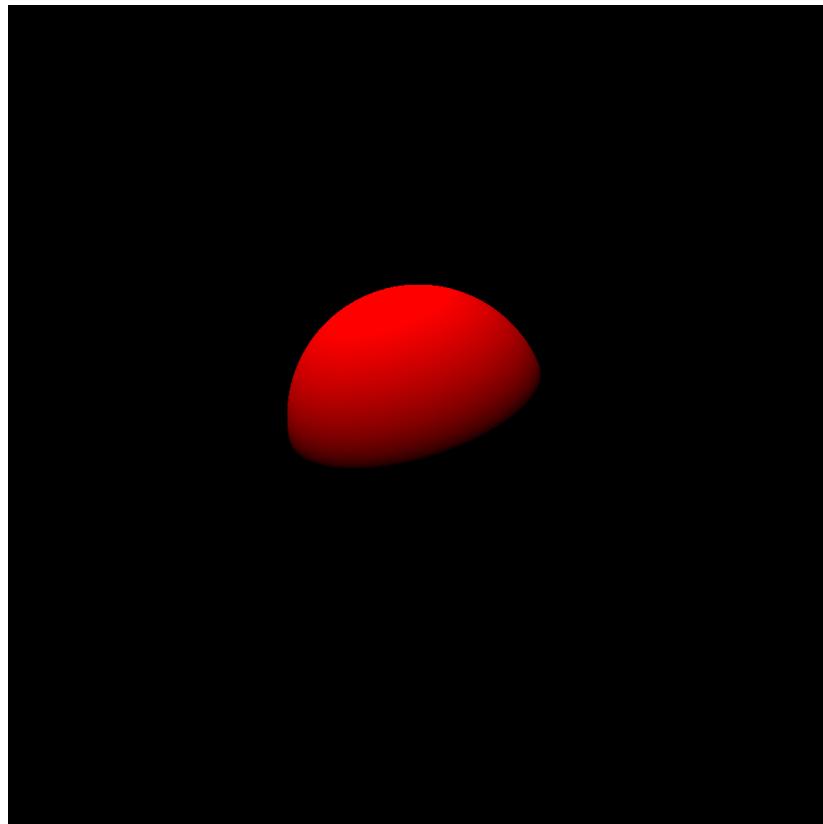


FIGURE 2 – Image de rendu 1

4.2 Ombre portée

Maintenant, on ajoute le sol, l'arrière-plan, un mur droit et gauche sous forme de grosse sphères avec des rayons très importants.

En plus de ces sphères, on va ajouter l'ombre de la petite sphère centrale. Pour ce faire, lorsque on trouve une intersection P , on génère un deuxième rayon de lumière partant de P et se dirigeant vers la source de lumière L (qui est pour le moment ponctuelle), et on cherche une seconde fois les intersections avec les éléments de la scène. S'il y a une intersection, on détermine le nouveau point d'intersection P' : si la distance de P à P' est inférieure à la distance de P à L , le point est ombré et sa couleur est donc noire $(0, 0, 0)$. Pour illustrer :

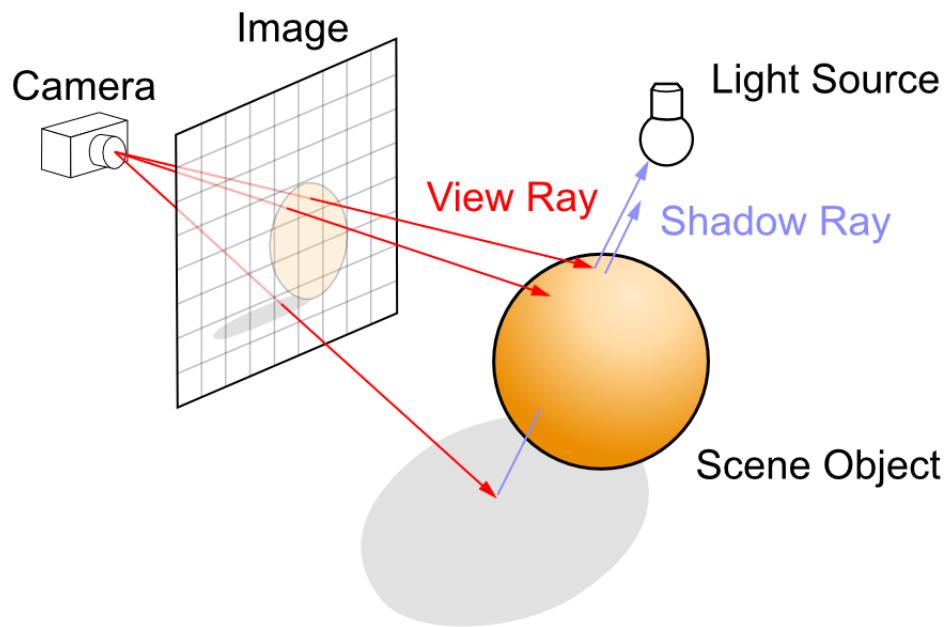


FIGURE 3 – Principe de l'ombre portée

l'image obtenu est très bruitée avec des tâches noires partout dans l'image. Ce bruit est dû à des imprécisions numériques. Pour résoudre ce problème, On peut lancer le rayon secondaire depuis un point légèrement décollé de la surface, en d'autres termes depuis le point : $P + \epsilon * n$ avec n : la normale à la surface en P . On obtient ainsi l'image suivante :

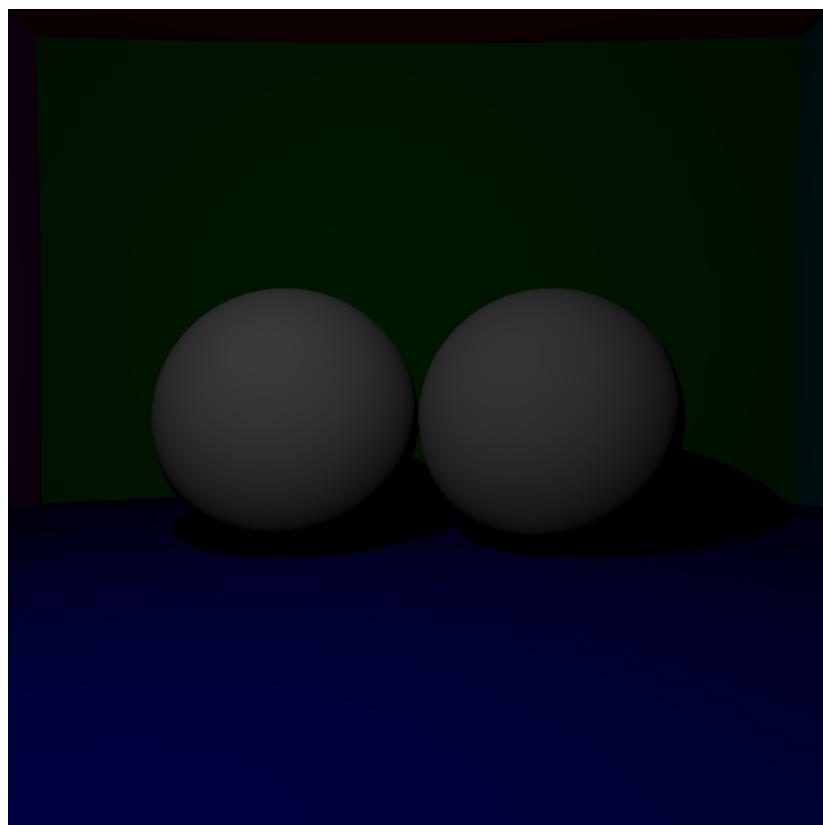


FIGURE 4 – Image de rendu 2

4.3 Surface miroir

Pour l'éclairage des surfaces de miroir on se base sur la réflexion totale comme suit :

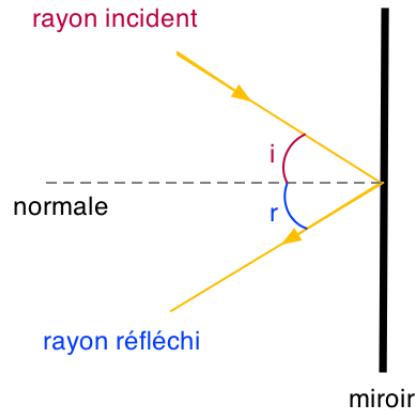


FIGURE 5 – Réflexion de la lumière sur une surface miroir

La fonction de réflexion prend un rayon incident, et change sa direction afin qu'il soit réfléchi. Cela est donné par la formule :

$$\vec{r} = \vec{i} - 2 < \vec{i}, \vec{n} > \vec{n} \quad (18)$$

avec \vec{i} est le vecteur incident et \vec{n} la normale à la surface.

L'image obtenue est la suivante :

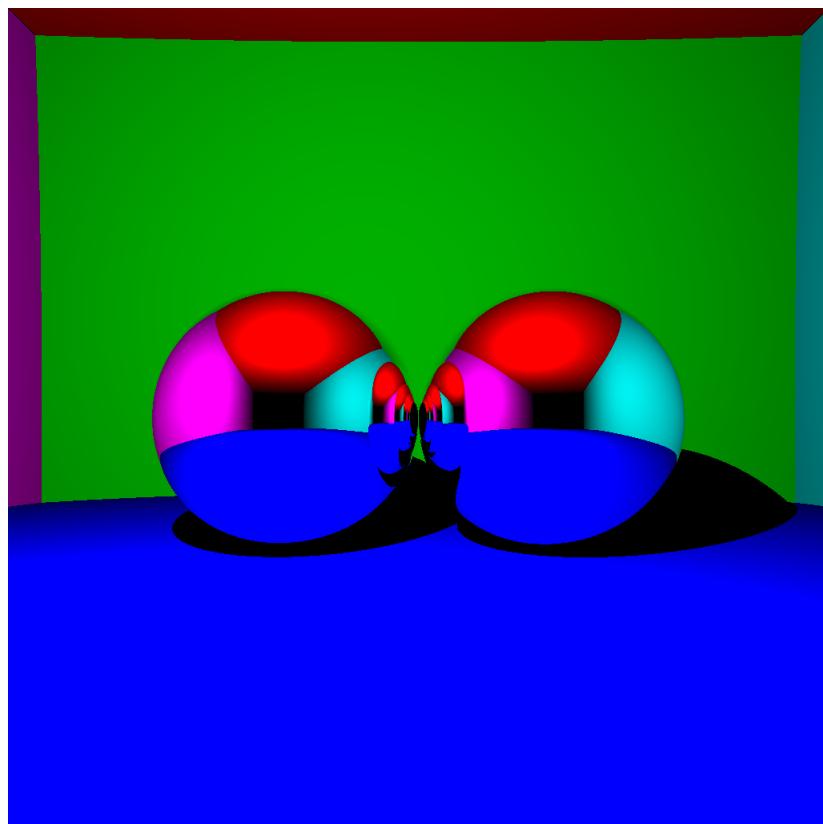


FIGURE 6 – Image de rendu 3

4.4 Surface transparente

Un rayon qui arrive sur une surface transparente sera réfracté. La formule permettant de calculer le rayon réfracté à partir d'un rayon incident à la surface est issu de la loi de Snell-Descartes : $n_{air} \sin \theta_i = n_{objet} \sin \theta_r$ où les n représentent les indices de réfraction de l'air et de la sphère.

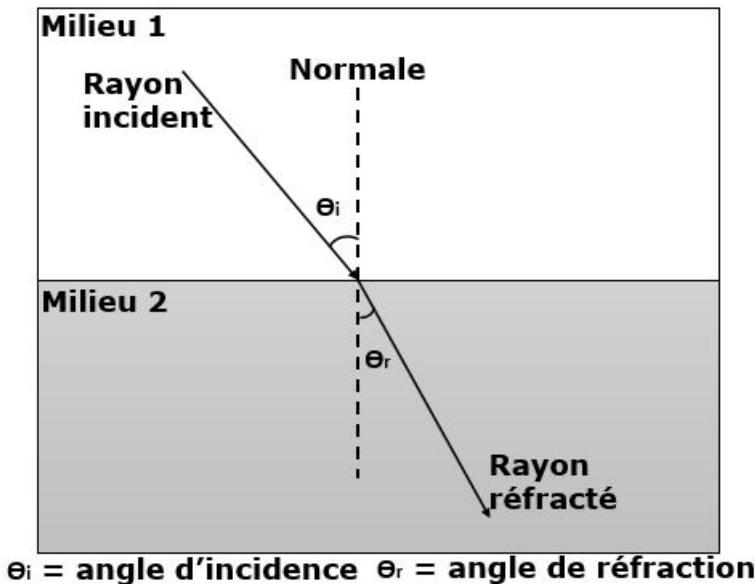


FIGURE 7 – Réfraction de la lumière

Si on considère \vec{T} le rayon transmis et que $\vec{T} = \vec{T}_t + \vec{T}_n$ où \vec{T}_t est la composante tangentielle et \vec{T}_n la composante normale, on obtient d'après la loi de Snell-Decartes :

$$\vec{T}_t = \frac{n_{air}}{n_{sphere}} (\vec{i} - \langle \vec{i}, \vec{n} \rangle \vec{n}) \quad (19)$$

$$\vec{T}_n = \sqrt{1 - \left(\frac{n_{air}}{n_{sphere}}\right)^2 (1 - \langle \vec{i}, \vec{n} \rangle^2)} \vec{n} \quad (20)$$

Remarques :

- Si la racine est complexe, cela veut dire qu'il n'y a pas de transmission, et la réflexion est totale.
- Les intensités lumineuses affichées à l'écran ne varient pas linéairement avec les valeurs données dans l'algorithme. En fait, les écrans ont une puissance γ ($luminosit = intensit^\gamma$, où $\gamma = 2.2$). Alors on applique la fonction $x^{1./2.2}$ sur chaque composante rouge, verte et bleue des couleurs de pixels.

On obtient ainsi l'image de rendu suivante :

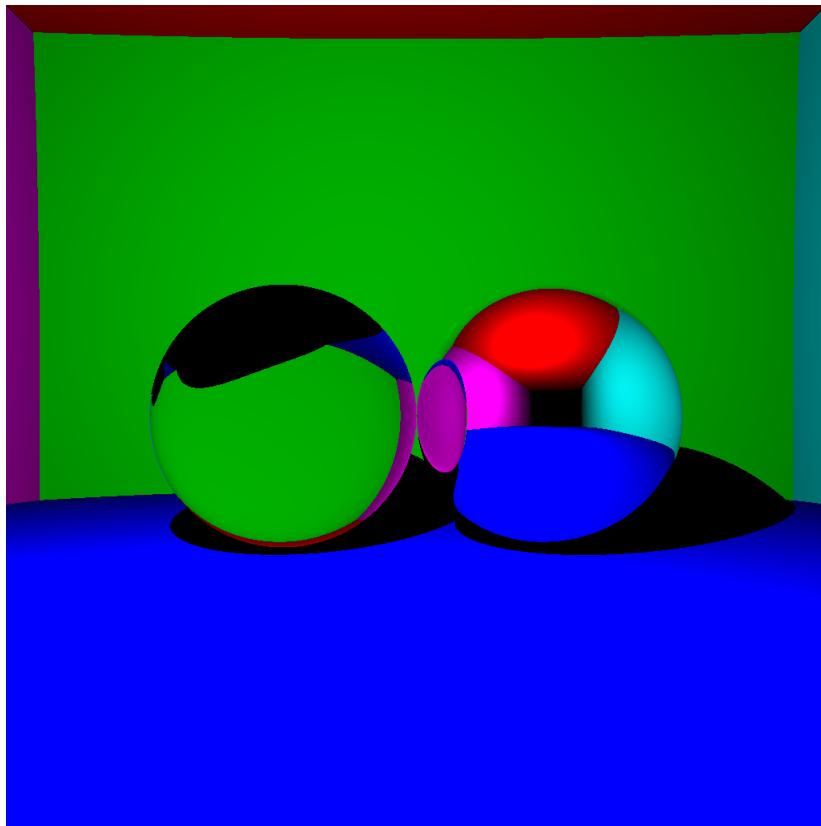


FIGURE 8 – Image de rendu 4

4.5 Éclairage étendue

Pour l'éclairage étendue, on transforme la source lumineuse L dans la classe Scène en une sphère avec un rayon et un origine au lieu d'un simple point (Vector). Maintenant, on ne considère que les rayons qui arrivent et intersectent cette sphère. Donc plus le rayon de la sphère de lumière est grand plus on a plus de probabilité qu'un rayon lancé arrive à cette source.

On peut aussi ajouter plusieurs sphères et/ou triangles dans la scène. (Quelques exemples d'images de la scène sont présentées dans le paragraphe suivant).

4.6 Éclairage indirect

Pour ajouter la contribution de l'éclairage indirect dans la fonction de calcul de couleurs des pixel "getColor()", on utilise l'équation locale suivante :

$$L_o(x, \vec{o}) = E(x, \vec{o}) + \int f(\vec{i}, \vec{o}) L_i(x, \vec{i}) \cos\theta_i d\vec{i} \quad (21)$$

où $E(x, o)$ est l'émissivité de la surface, $f(\vec{i}, \vec{o})$ est la BRDF, $L_i(x, \vec{i})$ l'intensité lumineuse arrivant au point d'intersection x , $L_o(x, \vec{o})$ l'intensité lumineuse qui en sort, $\cos\theta_i$ est le cosinus de l'angle entre le rayon incident \vec{i} et la normale à la surface.

Pour approximer l'intégrale, on utilise la méthode de Monte-Carlo : Il s'agit d'une méthode stochastique pour calculer des intégrales d'une manière approximative. On prend des

échantillons (x_0, x_1, \dots, x_n) aléatoirement de une fonction f suivant une loi de probabilité $p(x)$. On a :

$$\int f(x)dx = \frac{1}{n} \sum_{i=0}^n \frac{f(x_i)}{p(x_i)} + O(\sqrt{n}) \quad (22)$$

En pratique, plus la distribution p est proche de f , plus la méthode sera précise. Dans notre cas, afin de calculer l'intégrale de l'équation 21 pour les surfaces diffuses, il s'agit d'échantillonner uniquement le facteur en $\cos\theta_i$ puisque la BRDF est constante et connue : on génère des coordonnées x, y, z telles que :

$$x = \cos(2\pi r_1)\sqrt{1 - r_2} \quad (23)$$

$$y = \sin(2\pi r_1)\sqrt{1 - r_2} \quad (24)$$

$$z = \sqrt{r_2} \quad (25)$$

où r_1 et r_2 sont des nombres aléatoires uniformes entre 0 et 1. Il faut ensuite orienter ce rayon dans l'axe de la surface intersectée. Donc on applique une simple formule de changement de repère de ce rayon vers un nouveau repère qu'on le construira en générant des vecteurs perpendiculaires à la normale de la surface.

Pour améliorer aussi l'image, on fait un lissage des bordures des sphères en lançant non seulement un rayon au centre de chaque pixel mais plusieurs rayons par pixel dans différents parties de pixel. On obtient ainsi l'image suivante :

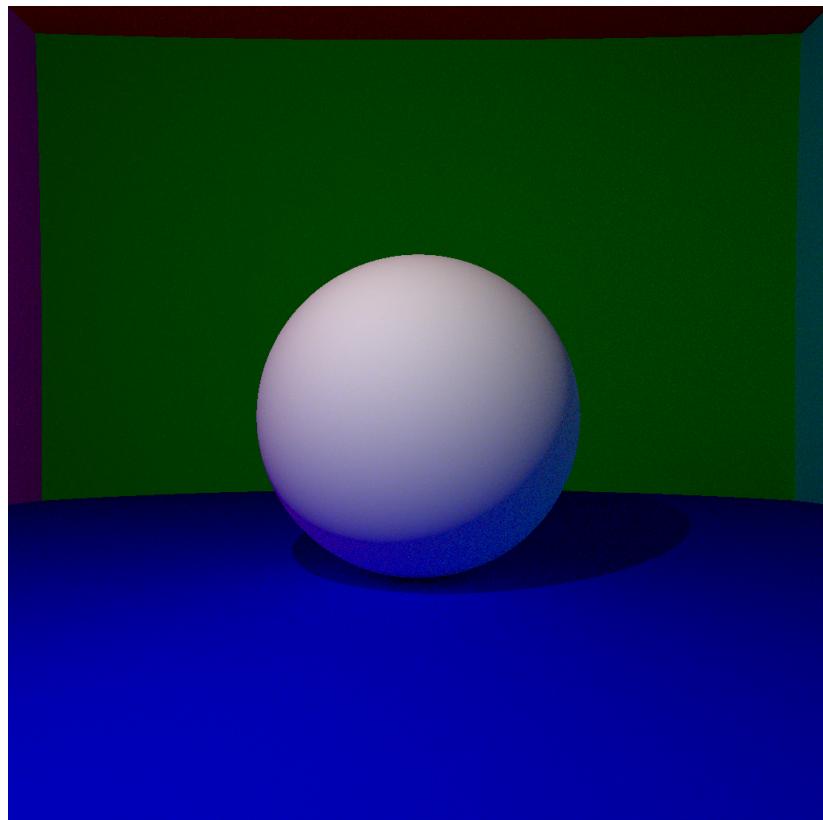


FIGURE 9 – Image de rendu 5

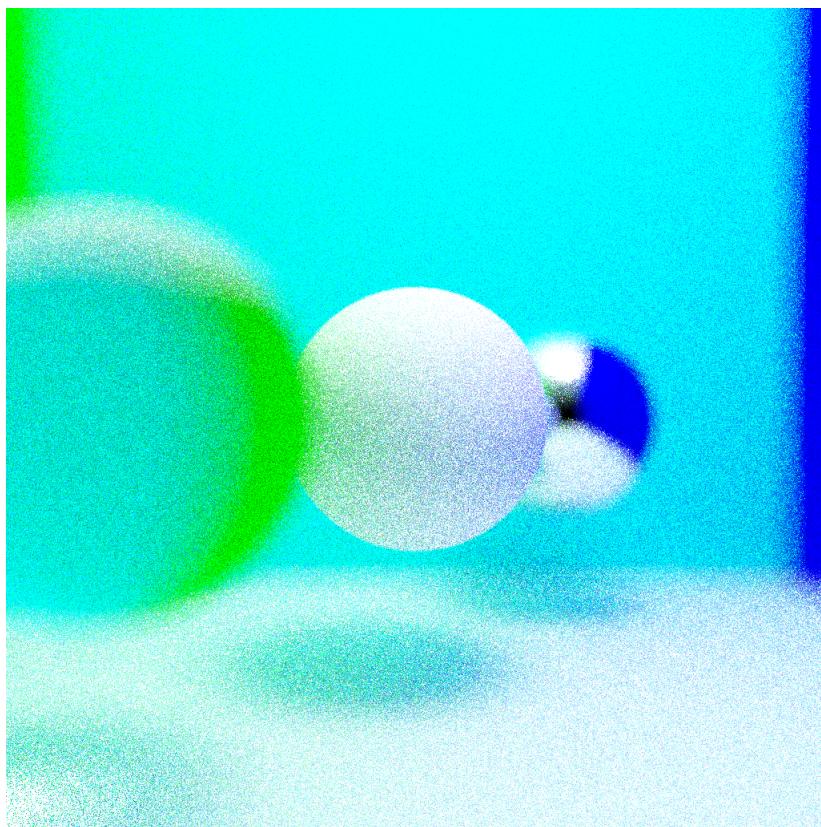


FIGURE 10 – Image de rendu 6 avec 50 rayons par pixel

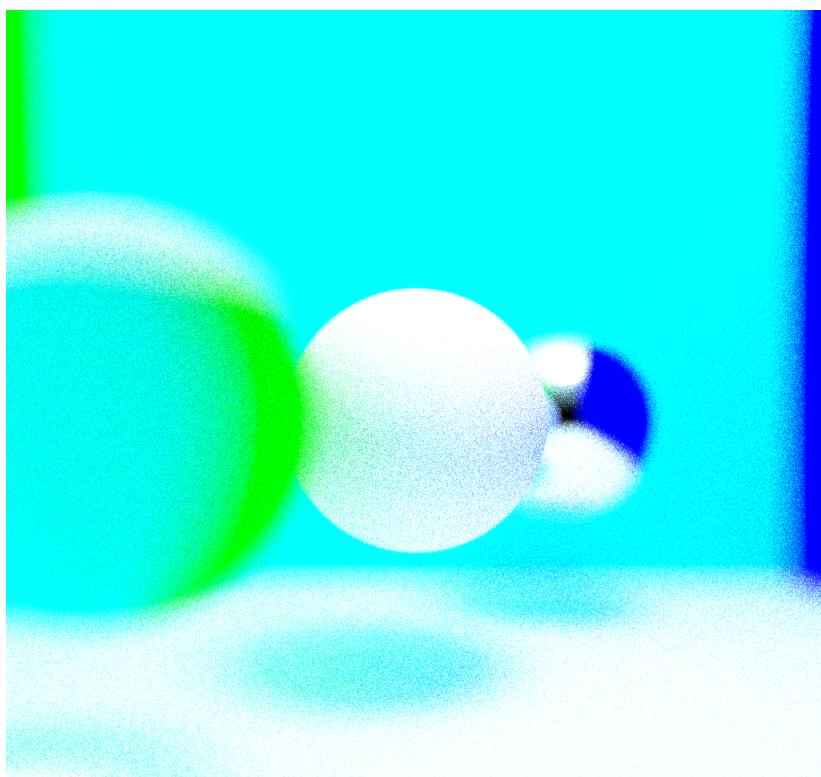


FIGURE 11 – Image de rendu 7 avec 100 rayons par pixel

On remarque qu'en augmentant le nombre de rayons lumineux lancés le bruit dans l'image de rendu diminue et l'image devient plus claire.

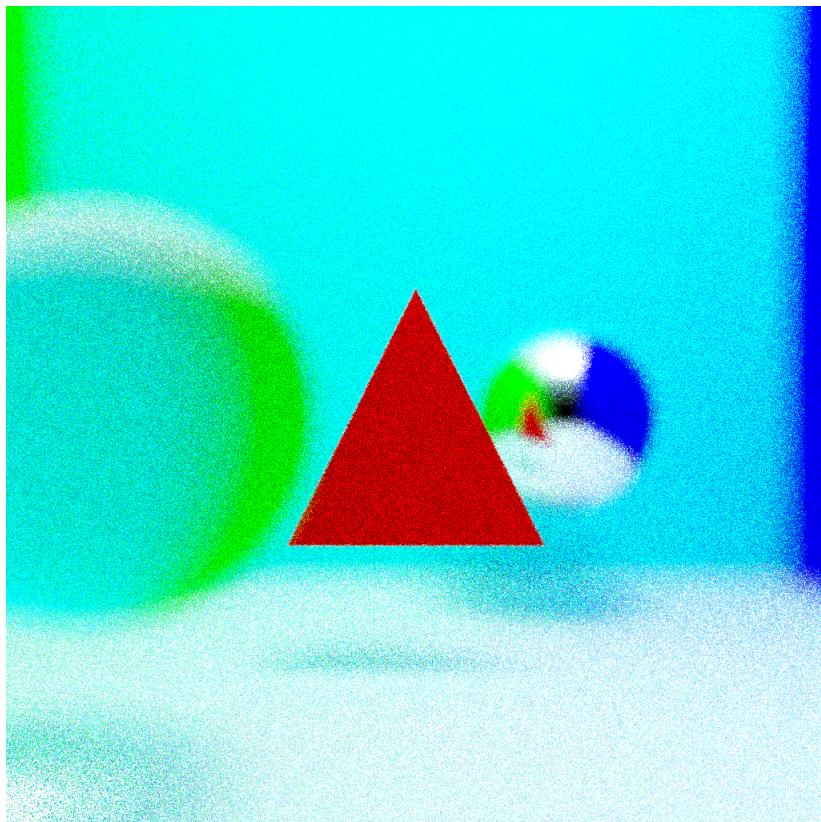


FIGURE 12 – Image de rendu 8

4.7 Le maillage

Pour le maillage, on utilise des fichiers ".obj" qui sont faciles à lire et à charger. Il s'agit d'un fichier où on stock les différents sommets des différents facettes triangulaires qui constituent l'objet complexe et leur vecteurs normales. On peut générer une boîte englobante "BBox" à cet objet pour limiter le nombre de rayon (on considère que les rayon qui intersecte la boîte) et enfin on peut calculer les intersections des rayons lumineux avec l'objet en utilisant la classe "Triangle" qui a comme sommet les sommets lus fichier ".obj".

On obtient l'image suivant :

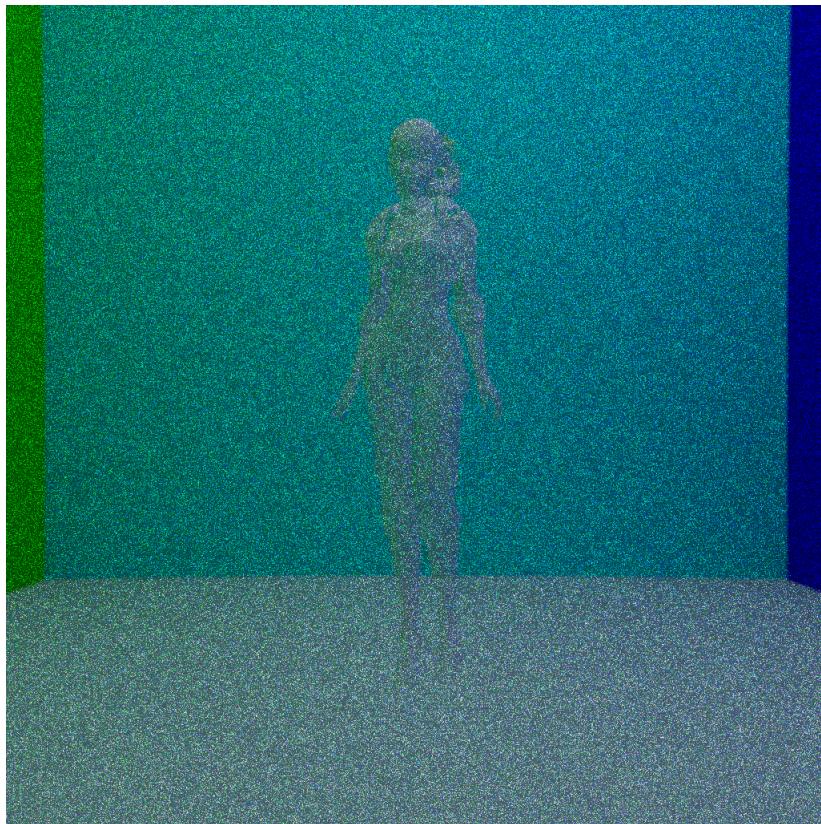


FIGURE 13 – Image de rendu 9

l'image est un peu bruité. Il faut augmenter encore le nombre des rayons lancés pour minimiser ce bruit mais cela prend beaucoup de temps de calcul. Pour cela on peut accélérer encore plus l'algorithme, en ajoutant l'algorithme "BVH" qui découpe l'objet en plusieurs boîtes et sous-boîtes en commençant par la boîte englobante comme racine. Chaque boîte englobe un ensemble bien déterminé de facettes triangulaires de l'objet. Cela nous donne une arborescence bien ordonnée de boîtes englobantes. Grâce à cet arbre de boîtes, on peut localiser très rapidement (en faisant un parcours en profondeur de la racine vers les feuilles de l'arbre) le triangle (i.e facette) qui intersecte un rayon lumineux donné. On obtient ainsi la même image mais en beaucoup moins de temps de calcul ce qui permet d'augmenter le nombre de rayon et minimiser le bruit pour avoir une image plus nette et claire. Remarque : mon ordinateur est un peu ancien son processeur et sa carte graphique n'ont pas de grande puissance de calcul c'est pour cela la génération des images était avec le minimum de rayons possible.

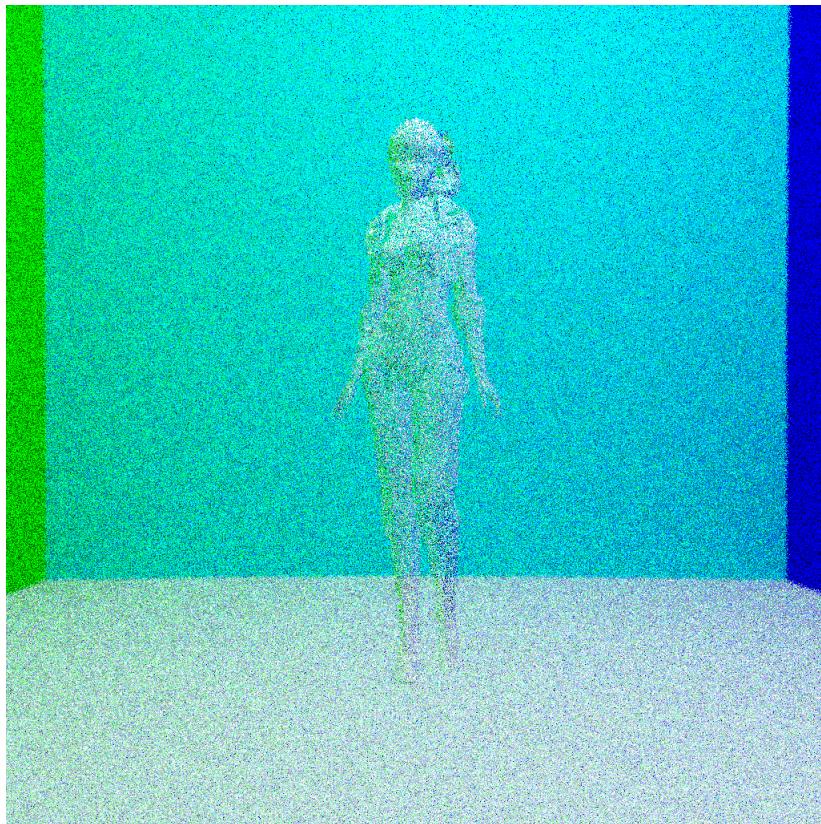


FIGURE 14 – Image de rendu 10

5 Conclusion

Dans ce TP, nous avons appris quelques techniques de base dans le raytracing et comment utiliser ces techniques pour simuler une scène en 3D en utilisant aussi la géométrie et les lois de la physique (optique et interaction lumière-matière). Nous avons simulé des différents objets avec des différentes surfaces (miroir, transparente, diffuse). Cette technique de raytracing est très efficace et précise dans la simulation des scènes 3D. Elle donne des résultats réalistes mais elle consomme beaucoup de ressources et prend beaucoup de temps de calcul surtout si on fait pas l'optimisation des algorithmes : Ce qui la rend inadéquate dans les applications temps réel.