



ÉCOLE  
**CENTRALE**LYON

# ÉCOLE CENTRALE LYON

MOS 2.2  
INFORMATIQUE GRAPHIQUE

## Raytracing

*Elève :*  
Jean WOLFF

*Enseignant :*  
Nicolas BONNEEL

Dans le cadre du cours d'informatique graphique en dernière année à l'École Centrale de Lyon, on implémente un raytracer. Les objets visualisés seront initialement uniquement des sphères, puis nous ajouterons les triangles et maillages, ainsi que de nombreux effets visuels, pour obtenir un rendu de plus en plus réaliste.

### Logiciel et compilateur

Le projet a été réalisé avec XCode (sur macOS). La compilation s'effectuait dans le terminal, avec les commandes :

- Pour la compilation : `clang++ -Xpreprocessor -fopenmp main2.cpp Vector.cpp Classes.cpp -std=c++11 -fomp -O3`
- Pour l'exécution : `./a.out 5 image11.png 1024 1024`, où les paramètres sont successivement : le nombre de rayons par pixel, le nom de l'image, sa largeur et sa hauteur.

### Durée d'exécution

La compilation durait environ 5 secondes. L'exécution, durait généralement entre 0.1 s et quelques secondes. Toutes les durées d'exécutions étaient mesurées, mais seules celles supérieures à une dizaine de secondes sont précisées dans ce rapport. Le processeur est Intel Core i5 double cœur, et les calculs pour les différents pixels sont parallélisés grâce à openmp.

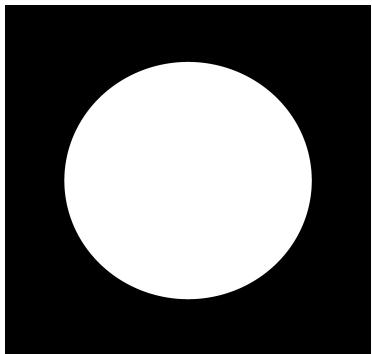
### Fonctionnalités implémentées :

- Intersection rayon - sphère
- Éclairage d'un matériau diffus
- Ombres portées
- Surfaces spéculaires
- Surfaces transparentes
- Sphère creuse
- Éclairage indirect avec intégration de Monte Carlo
- Anti-aliasing
- Lumières étendues
- Ombres douces
- Profondeur de champ
- Correction gamma
- Intersection rayon – triangle
- Ajout d'un maillage
- Accélération par intersection rayon – boîte englobante.
- Accélération par « bounding volume hierarchies ».
- Lissage de Phong
- Ajout d'une texture à un maillage
- Réflexion de Fresnel

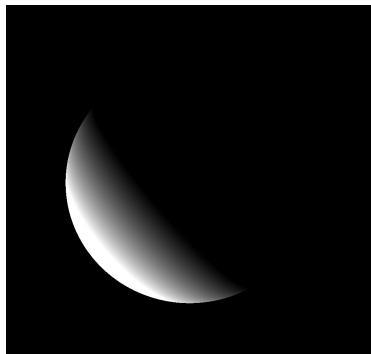
### Erreurs de rendu

Plusieurs erreurs de rendus ont été longtemps présentes dans la fonction `getColor`. Ces erreurs n'ont été corrigées qu'une fois le projet quasiment terminé, après l'ajout des textures.

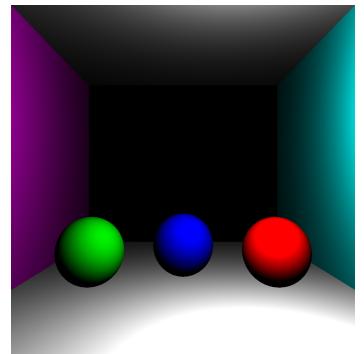
## I - Sphères, scène, ombre, surfaces spéculaires et transparentes



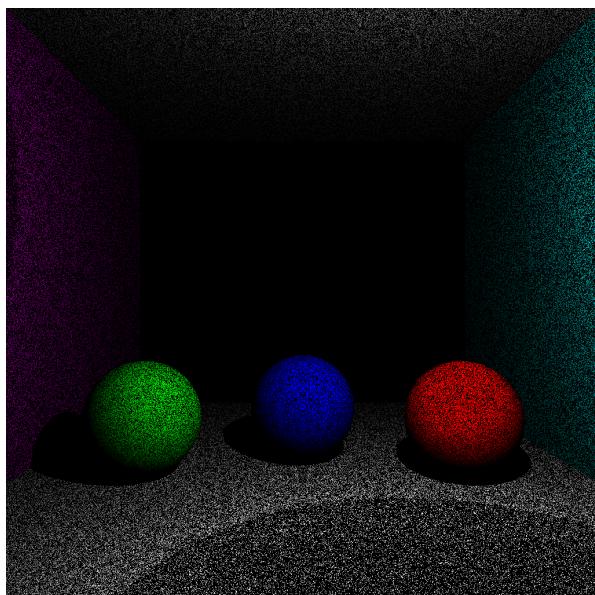
Intersection rayon -  
sphère



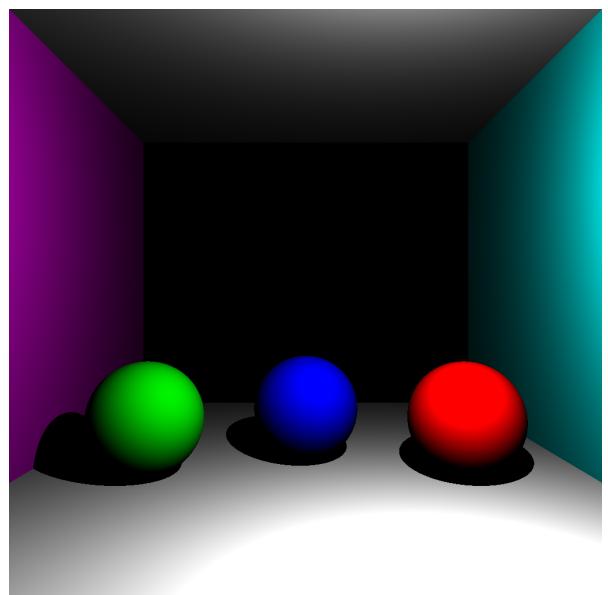
Eclairage d'un matériau  
diffus



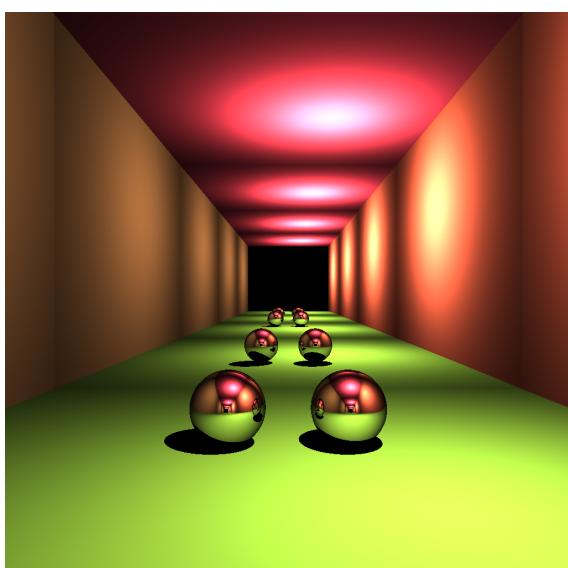
Intersection rayon -  
scène



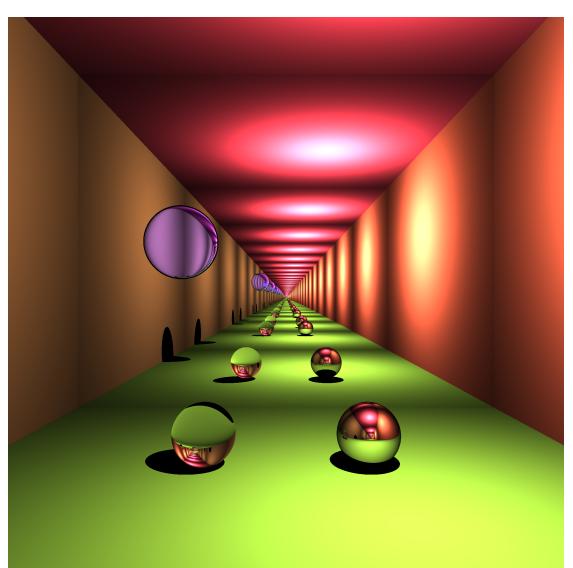
Ombres portées – bug du à l'imprécision  
numérique



Ombres portées

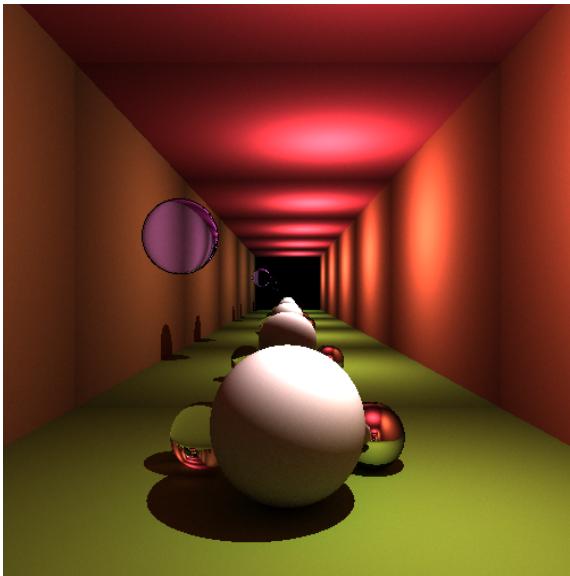


Surfaces spéculaires (5 rebonds au  
maximum)



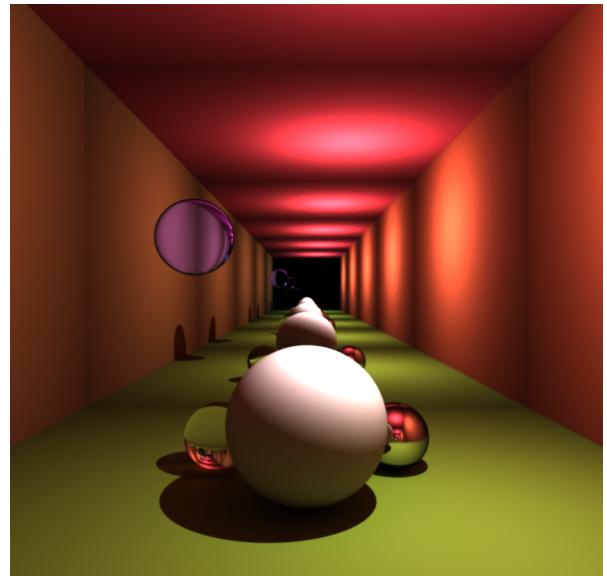
Surfaces transparentes et sphère creuse (30  
rebonds au maximum)

## II – Eclairage, anti-aliasing et profondeur de champ



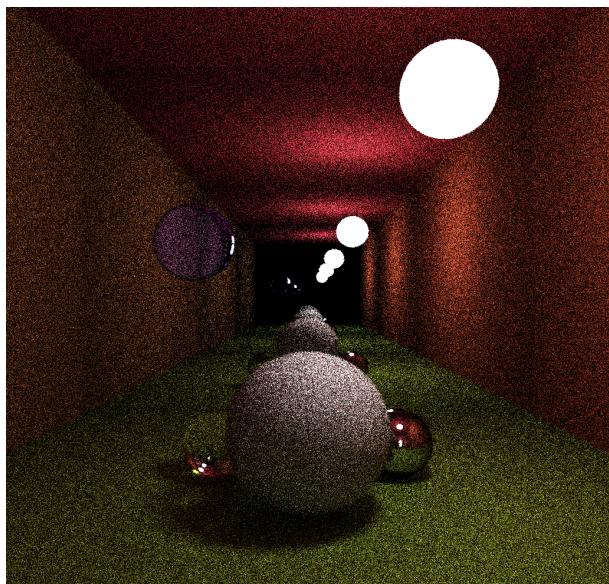
**Éclairage indirect avec intégration de Monte Carlo (imparfait, corrigé en dernière partie)**

Image 500x500, 100 rayons par pixel environ  
4 minutes de calcul.



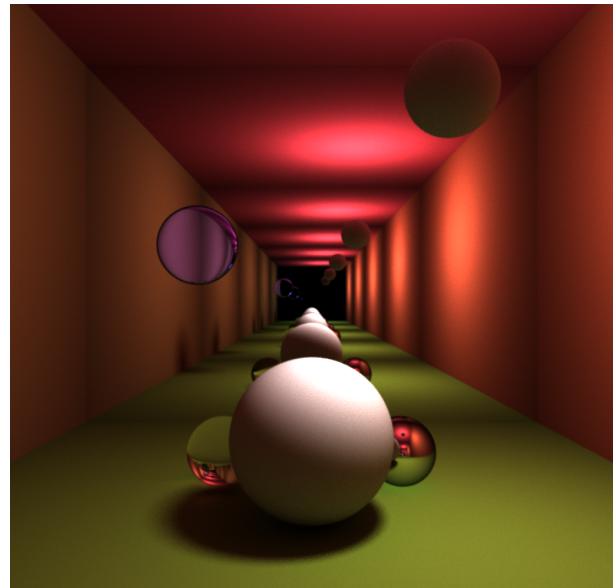
**Anti-aliasing**

Image 1024x 1024, 100 rayons par pixel,  
environ 5 minutes de calcul.



**Lumière étendue « naïve »**

Image 1024x1024, 100 rayons par pixel,  
environ 5 minutes de calcul.

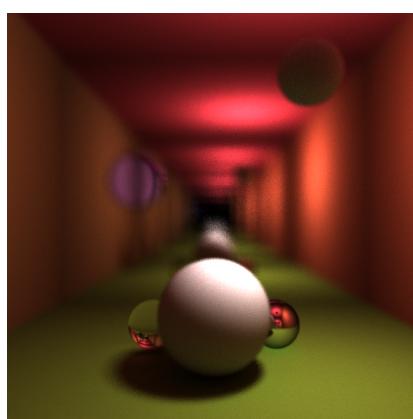


**Lumière étendue**

Image 1024x1024, 100 rayons par pixel,  
environ 5 minutes de calcul.

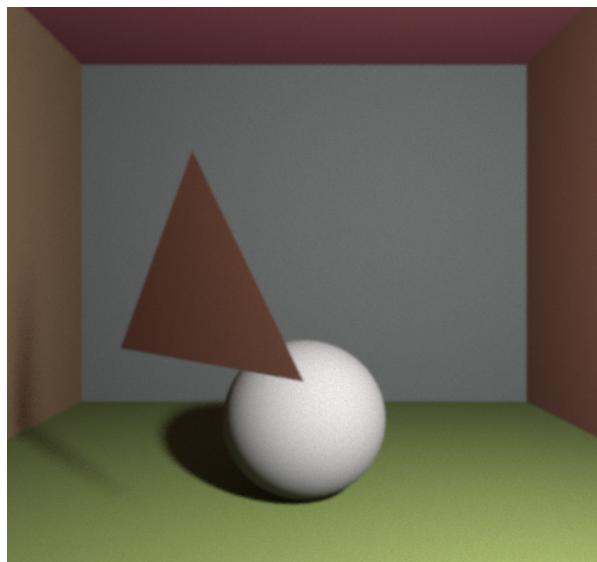
**Profondeur de champ**

Le plan net est celui des sphères miroir et transparente.



### III – Triangle, maillage et texture

La correction gamma a également été implémentée à cette étape. Un bug m'a fait refaire presque tout mon code sans vraiment observer les changements produits, avant de (enfin) trouver une coquille dans l'intersection de la classe scène (coquille qui ne posait pas de problème avec les sphères). Dans cette section, plusieurs erreurs dans getColor subsistent, l'éclairage indirect est désactivé, et l'anti-aliasing floute légèrement l'image. Ces erreurs sont corrigées en dernière partie. Toutes les images suivantes sont calculées avec un maximum de 5 rebonds.



**Ajout d'un triangle**

Image 500x500, 100 rayons par pixel, 28s de calcul.



**Maillage et Boite englobante**

Image 500x500, 15 rayons par pixel, 15min 55s de calcul.



**Accélération par BVH**

Image 700x700, 100 rayons par pixel, 20 min 50 s de calcul.

Une image de 500x500, 15 rayons par pixel, au lieu d'être calculée en environ 16 minutes comme précédemment, est maintenant calculée en seulement 55 secondes.



**Lissage de Phong**

Image 700x700, 100 rayons par pixel, 19min 39s de calcul.

### Ajout de textures

Image 700x700, 100 rayons par pixel, 7 min 30 s de calcul



## IV – Corrections

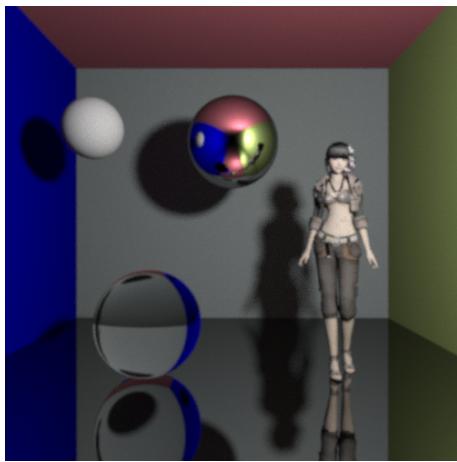
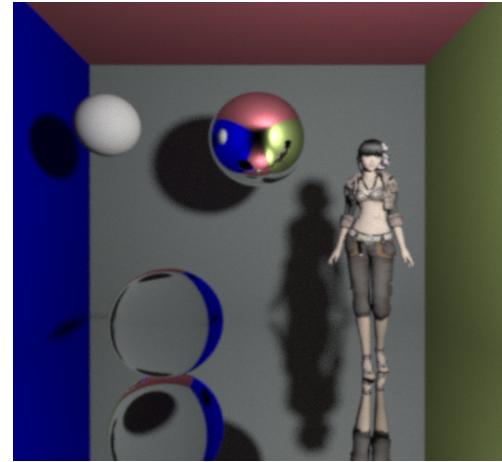


Image avant les corrections

Image 500x500, 100 rayons par pixel, 7 min 25 s



Correction des surfaces spéculaires dans getColor

Dans le cas d'un miroir, un produit scalaire était en trop :  
 $\text{couleur} = \text{dot}(\text{dir\_miroir}, \text{N}) * \text{getColor}(\text{ray\_miroir}, s, \text{reb}-1)$   
On a perdu la distinction entre le sol miroir et les murs.

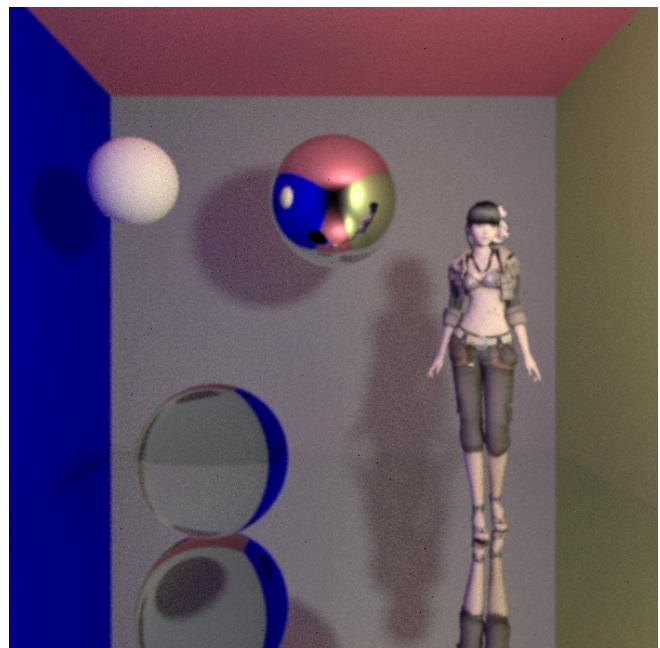
Image 500x500, 100 rayons par pixel, 7 min 38 s

### Correction d'erreurs de calcul dans getColor, pour les éclairages direct et indirect

On observe (enfin) une lumière indirecte correcte sur la sphère blanche, mais l'image est bruitée.

On récupère une fine distinction sol / murs.

Image 600x600, 100 rayons par pixel, 11 min



Ci-dessous l'une des erreurs : le  $\text{std::max}(0, \text{dot}(\text{Np}, -\text{wi}))$  a été corrigé en  $\text{dot}(\text{Np}, -\text{wi})$

```
// --- ECLAIRAGE DIRECT ---
else {
    double proba = dot(axe_P0, dir_aleatoire);
    intensite_pixel = intensite_lumiere / (4 * M_pi * d_light2) * std::max(0., dot(N, wi)) * std::max(0., dot(Np, -wi)) / proba * albedo;
}
```

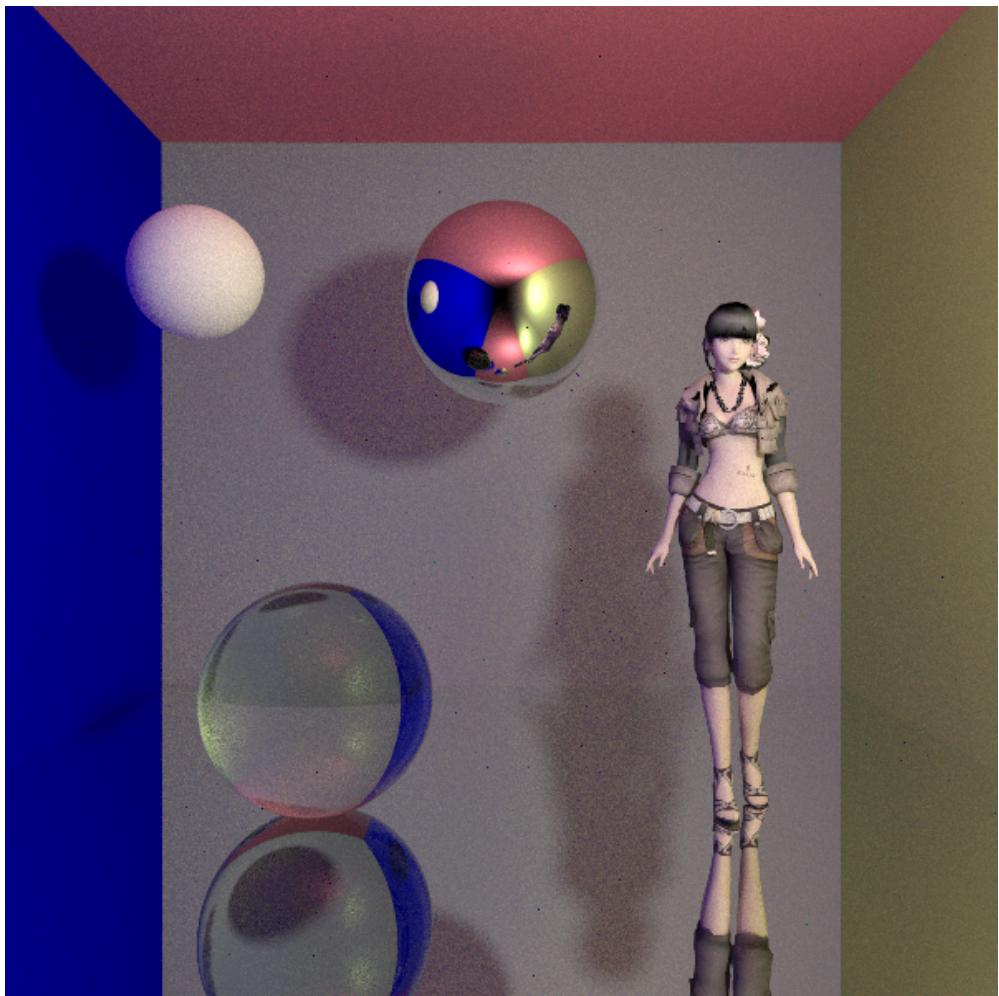
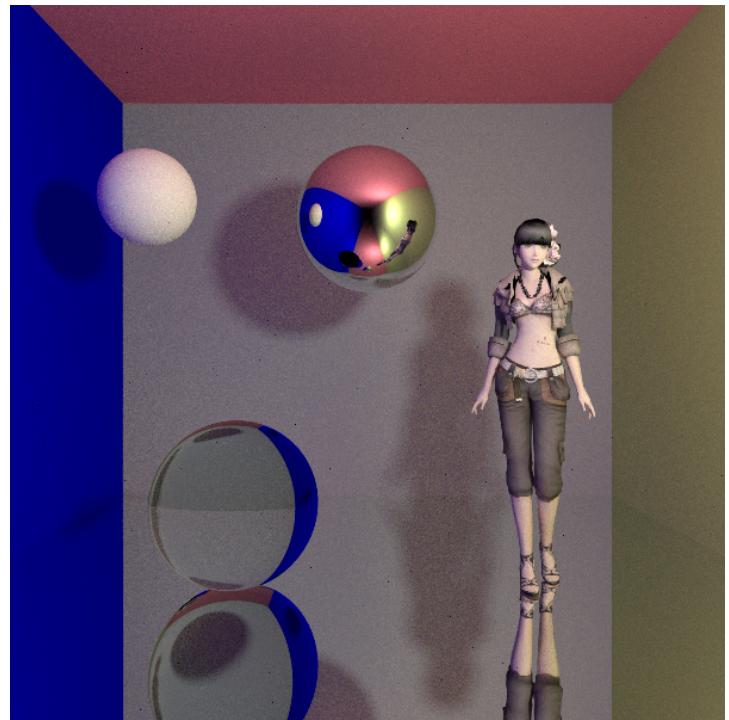
### Diminution de l'anti-aliasing

On multiplie dx et dy par un facteur 0.2, comme illustré ci-dessous (valeur 0.2 plus faible que la valeur « théorique », mais compromis entre flou et crénelage).

L'image gagne en netteté.

Image 600x600, 100 rayons par pixel, 10min36s

```
// Anti-aliasing - methode Box Muller
double r1 = uniform(engine);
double r2 = uniform(engine);
double R = 0.2 * sqrt(-2 * log(r1));
double dx = R * cos(2 * M_pi * r2);
double dy = R * sin(2 * M_pi * r2);
```



### Réflexion de Fresnel

Image 600x600, 100 rayons par pixel, 7min37s

## Retour sur le cours

L'un de mes meilleurs cours à Centrale : avoir un rendu visuel est très stimulant pour la programmation, on comprend bien ce qu'on fait, le professeur maîtrise très bien son sujet.

Ce que j'en ai retiré : la compréhension du modèle de raytracing. Je ne sais pas si cela me servira plus tard, mais c'était très intéressant. Et les bases de la programmation en C++.

A améliorer : envoyer un mail avant le premier cours pour dire aux étudiants d'installer un compilateur C++, et d'apprendre les bases du langage et du débogage. On est nombreux à avoir pris très vite beaucoup de retard car nous n'avions jamais fait de C++.