ÉCOLE CENTRALE LYON

MOS - Informatique Graphique

Rapport de Travaux Dirigés

Cyril LOME
Enseignant - Nicolas BONNEEL

Janvier - Mars 2019

Introduction

Les images de synthèse sont utilisées dans de multiples domaines (cinéma, jeux-vidéos, etc.). Il existe deux type de méthodes pour la synthèse d'image. En "rasterization", on projettera des formes géométriques à l'écran. Dans le cadre de ce cours, nous avons vu l'autre type de méthode, appelée "raytracing", consistant à envoyer des rayons dans une scène. Cette méthode est moins rapide que la première, mais permet d'obtenir des image beaucoup plus réalistes (en terme d'ombres, de textures, etc.).

Ainsi, nous avons pu, à travers plusieurs étapes, construire petit à petit un moteur de rendu en C++ qui fabrique des images de scènes définies en amont. Ce rapport sert à présenter les résultats obtenus (images, temps d'exécution) afin d'illustrer les performances du moteur de rendu que j'ai développé durant les séances dédiées.

I - Premiers rendus réalistes

Dans cette partie, nous allons présenter les résultats obtenus pour une lumière ponctuelle, du rendu d'une scène, avec les quelques implémentations suivantes :

1. **Calcul d'intensité à la surface d'une sphère**

On présente ici le premier rendu de scène perçu en 3D de notre travail. On trouve des sphères entourées par des "murs" (en réalité, des sphères de très grand rayon donnant cette impression de plan). Voir la Figure 1.

2. **Correction gamma**

A ce genre de rendu, on doit ajouter une correction gamma pour rendre plus réaliste le rendu sur un écran (cf. Figure 2).

3. **Ombres portées**

Les ombres portées s'ajoutent en étudiant les intersections des rayons en direction de la lumière avec une potentielle autre sphère. Si c'est le cas, nous ne mettons pour le moment que la couleur noir sur le pixel correspondant (cf. Figure 3).

4. **Sphère miroir**

Les sphères miroir reflètent tout simplement les rayons de manière totale, dans la direction miroir. On crée un nouveau rayon dans cette direction et la couleur du pixel au niveau de la sphère miroir sera celle que ce nouveau rayon retourne (avec un maximum de 5 rebonds, pour s'assurer qu'il n'y pas une boucle infinie en cas de plusieurs surfaces miroir) (cf. Figure 4).

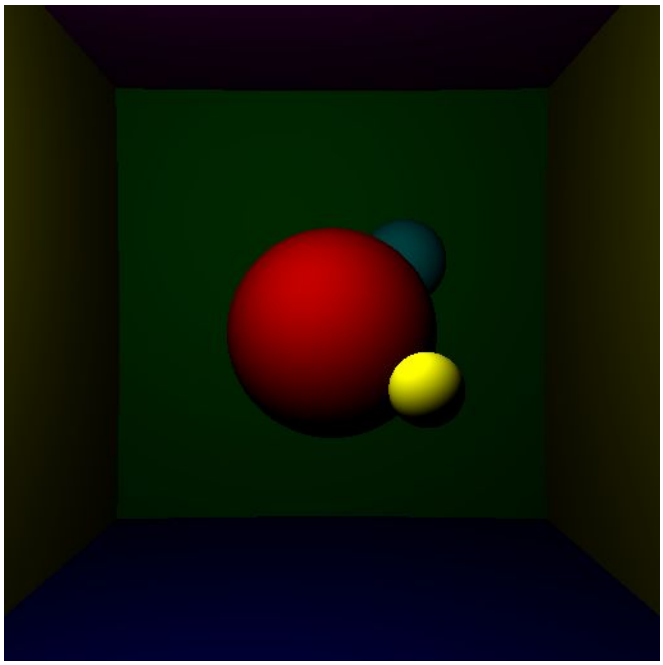

Figure 1 - Set de sphère de base

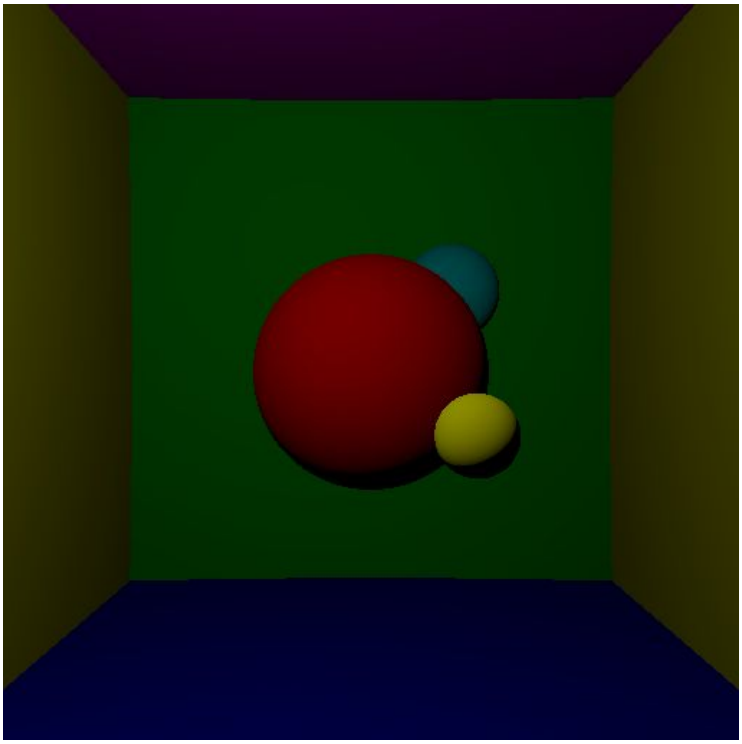

Figure 2 - Set de sphère de base avec correction gamma

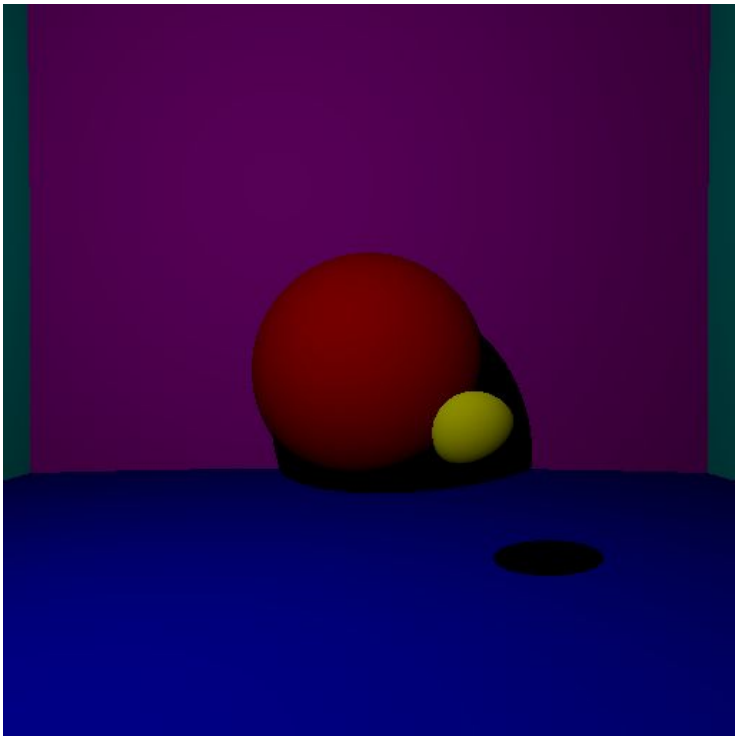

Figure 3 - Set de sphères avec ombres portées

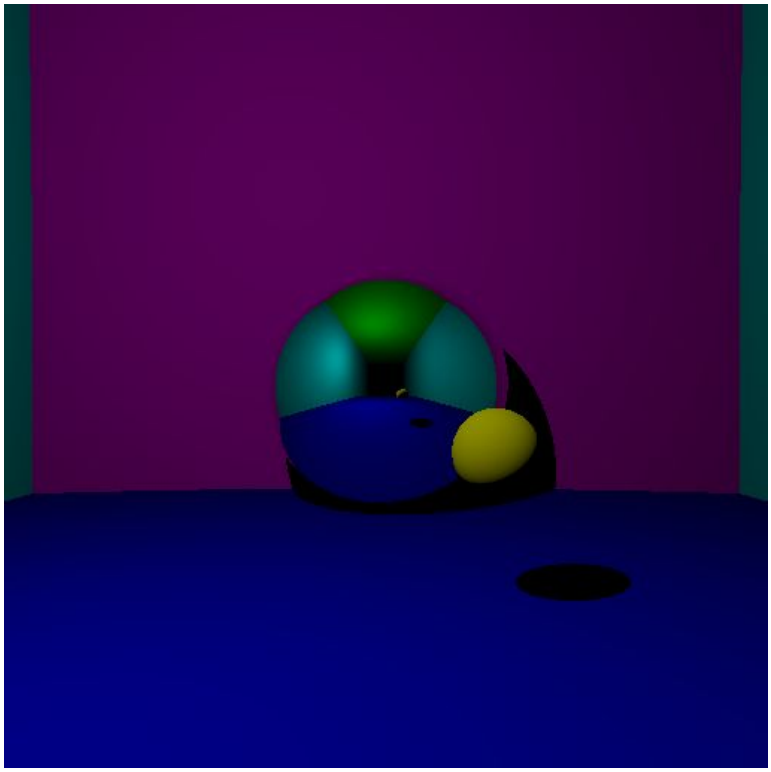

Figure 4 - Set de sphères avec une sphère miroir

II - Surface diffuses et antialiasing

Nous présenterons ici, à travers quelques images, les rendus obtenus pour des surfaces de sphère diffuses, en considérant un nombre croissant de rayons envoyés par pixels. Puis nous verront l'amélioration du rendu apportée par l'antialiasing.

1. Les surfaces diffuses

Afin d'obtenir un rendu plus réaliste des couleurs et des ombres, on ajoute une composante de lumière indirecte, issue de la réflexion de rayons sur la surface des sphères. Voir les Figures 5, 6 et 7.

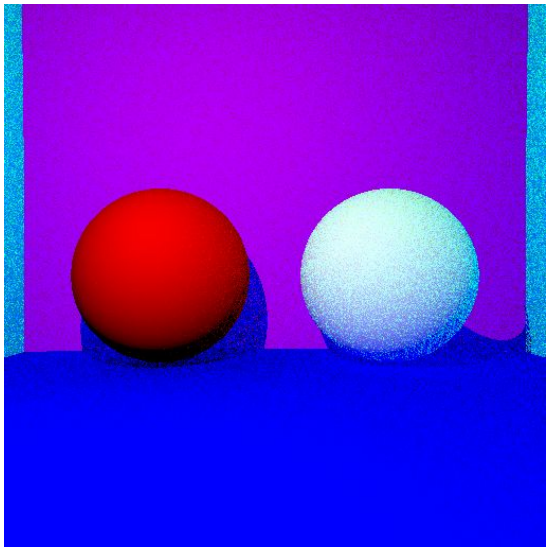

Figure 5- Set de sphère avec composante de lumière indirecte - 5 rayons

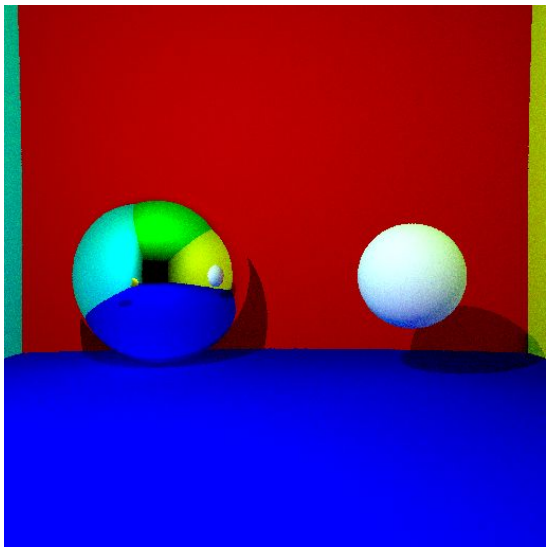

Figure 6 - Set de sphère avec composante de lumière indirecte - 50 rayons

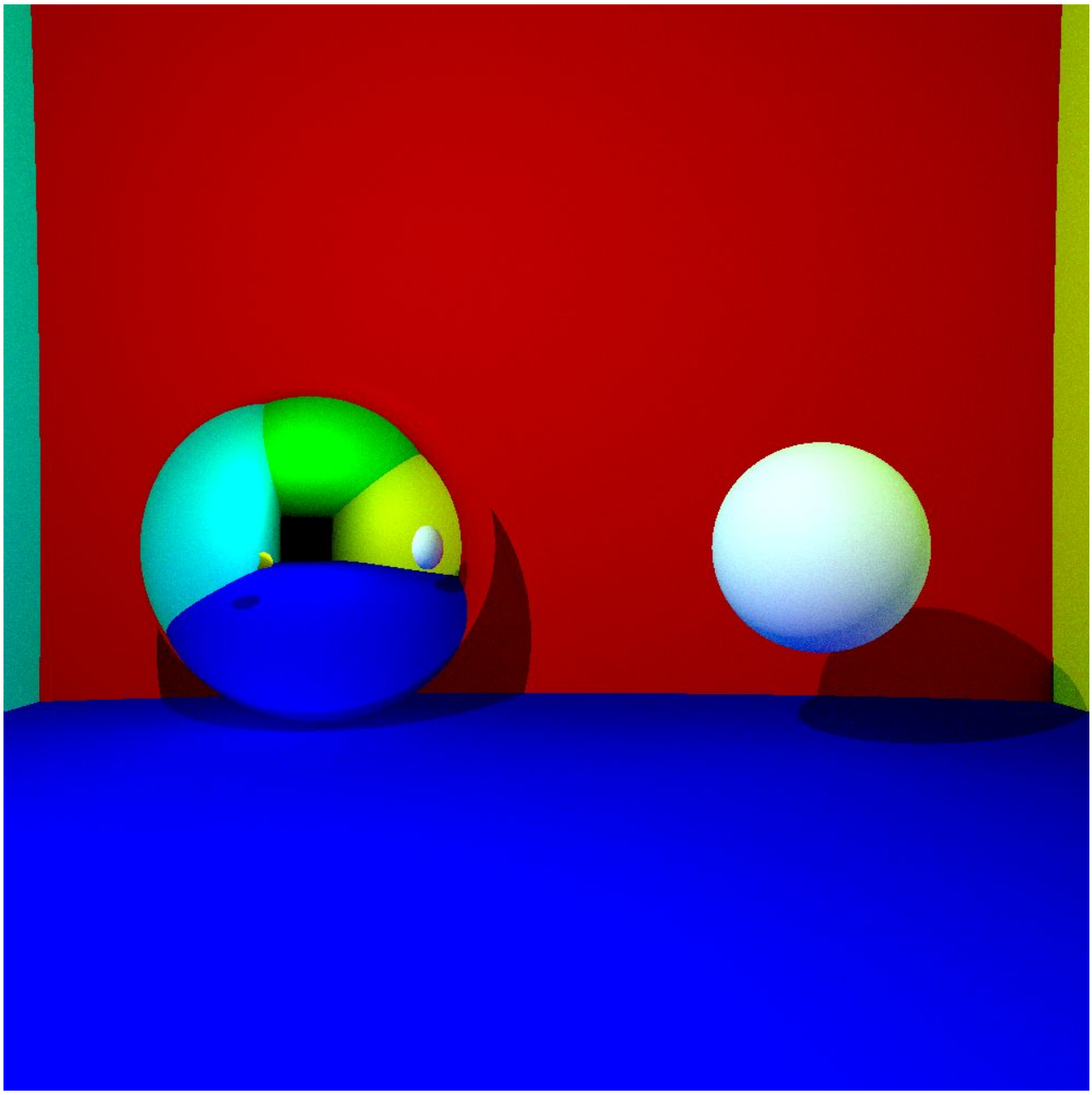

Figure 7 - Set de sphères avec composante de lumière indirecte - 100 rayons, grande taille - 2h d'exécution (pas de parallélisation)

2. L'antialiasing

Il s'agit d'une méthode permettant de lisser les contours qui, de base, ont tendance à être crénelés (comme on peut voir sur les Figures 8 zoomant sur les contours pour un rendu sans et avec antialiasing. La Figure 9 montre le rendu total avec antialiasing.

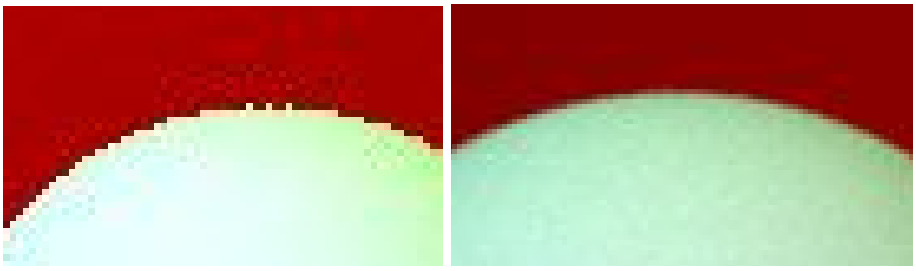

Figure 8 - A gauche, sans antialiasing / à droite, avec antialiasing

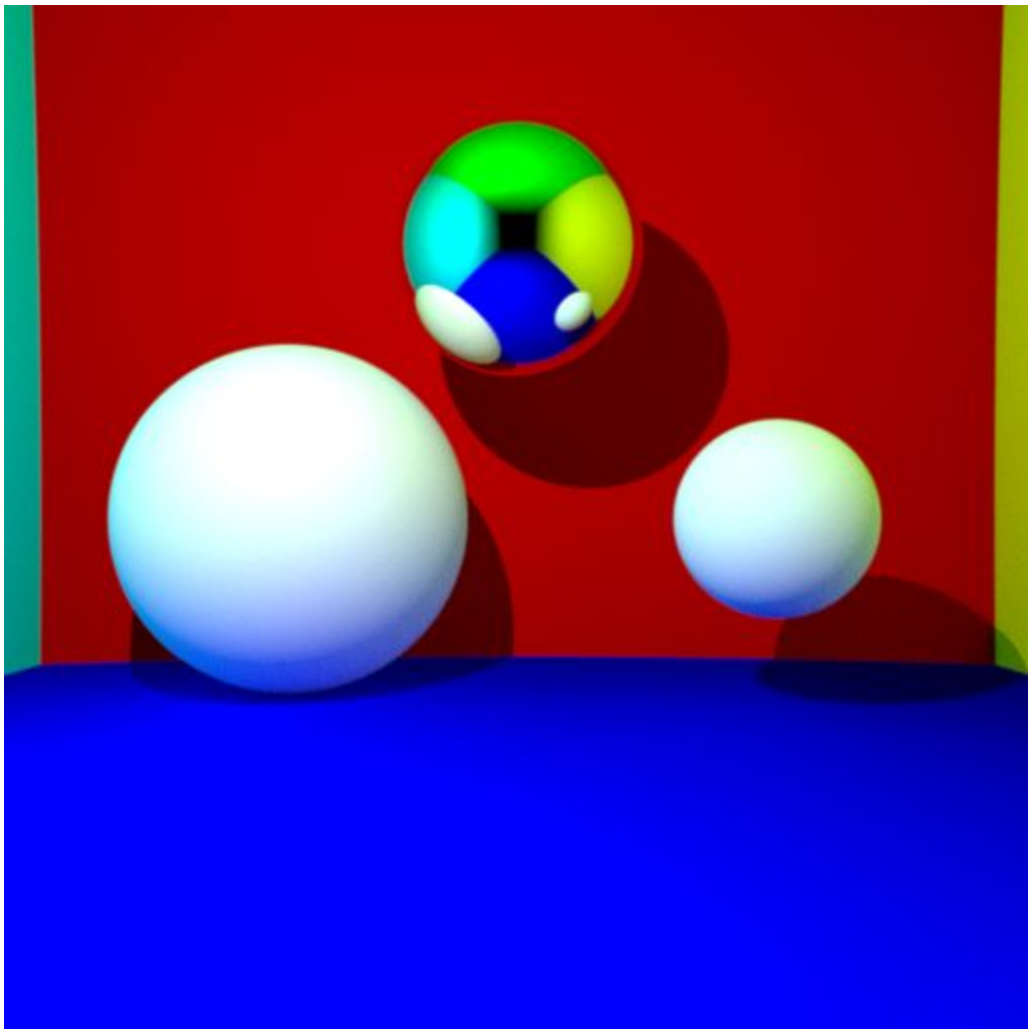

Figure 9 - Set de sphères avec lumière ponctuelle, composante de lumière indirecte et antialiasing - 1000 rayons - 5 min

III - Lumière sphérique

Dans cette partie, nous verrons les rendus obtenus avec une lumière non pas ponctuelle, mais sphérique. Cette approche permet d'obtenir des résultats encore plus réalistes en éclairage.

Dans notre cas, l'implémentation s'effectue de manière correcte, mais, pour une raison que je n'ai pas réussi à trouver, la composante de lumière directe ne calcule pas bien les endroits non éclairés par la source de lumière (car un autre objet fait obstacle). De cette manière, comme le montreront les images qui vont être présentées, les ombres n'apparaissent plus. Selon moi, il pourrait s'agir d'un bug situé dans la routine d'intersection ou bien d'une condition d'intersection mal écrite (bien que j'aie passé des heures sur cette partie du code et qu'elle me semble bel est bien logique et correcte).

On voit sur les Figures 10 et 11 que lorsque le nombre de rayons augmente, la qualité augmente, mais comparé à la lumière ponctuelle, il faut un nombre bien plus conséquents de rayons pour atténuer ce bruit.

La Figure 1 permet de voir le rendu final acceptable (sauf bug des ombres) de cette configuration de moteur de rendu, pour 5000 rayons, avec une sphère lumineuse relativement peu éloignée, d'un rayon de 5.

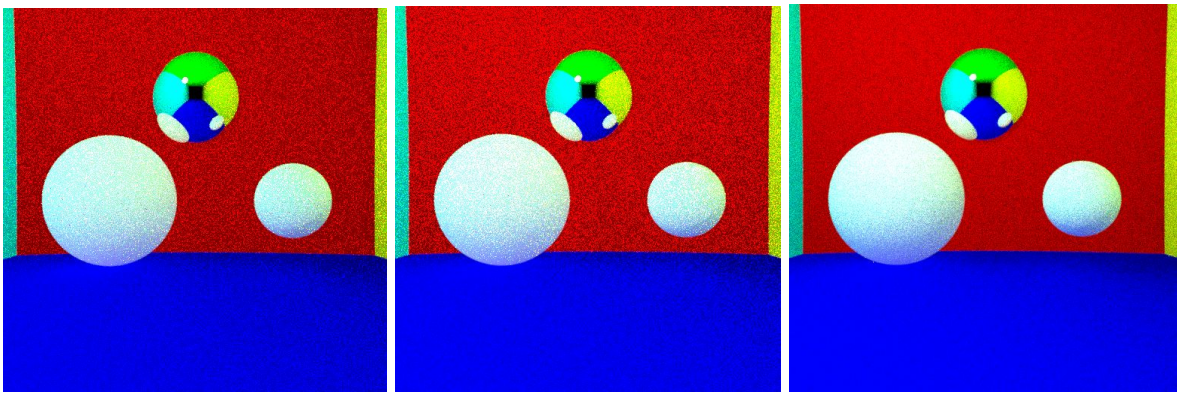

Figure 10 - Scène par lumière sphérique pour un nombre de rayons de 50, 100 et 1000

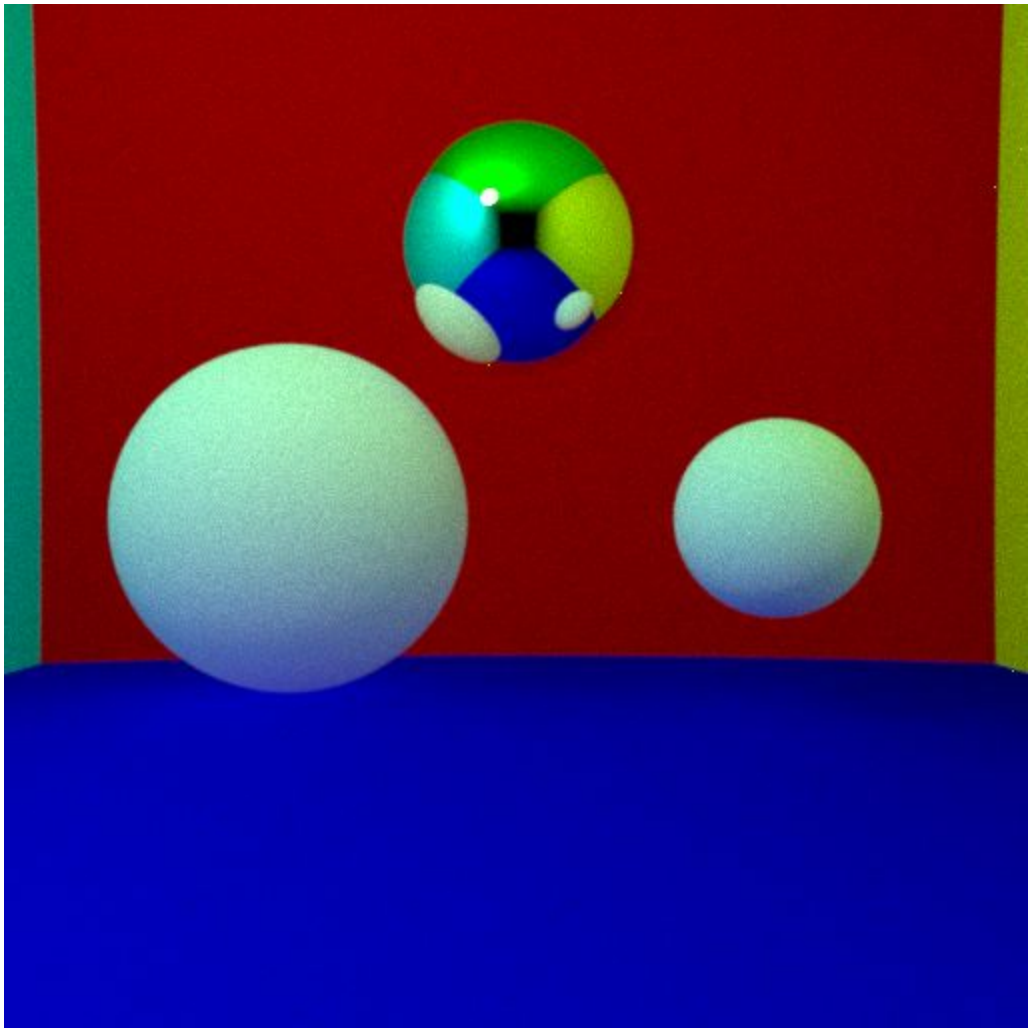

Figure 11 - Set de sphères avec composante de lumière sphérique - 5000 rayons, grande taille - 20 min d'exécution

Enfin, les Figures 12 et 13 permettent de comparer, pour la même configuration, l'influence de la taille de la sphère lumineuse. On voit que la luminosité augmente avec la taille (énergie lumineuse émise totale moins élevée) ainsi que la qualité du rendu diminue (on a besoin de moins de rayons).

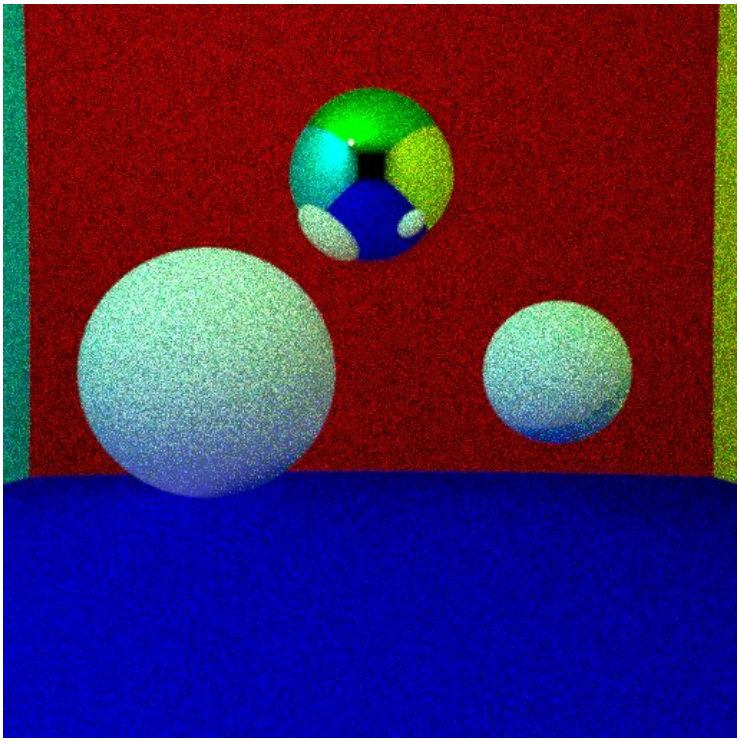

Figure 12 - Set de sphères avec composante de lumière sphérique de rayon 1 - 5000 rayons, grande taille - 20 min d'exécution

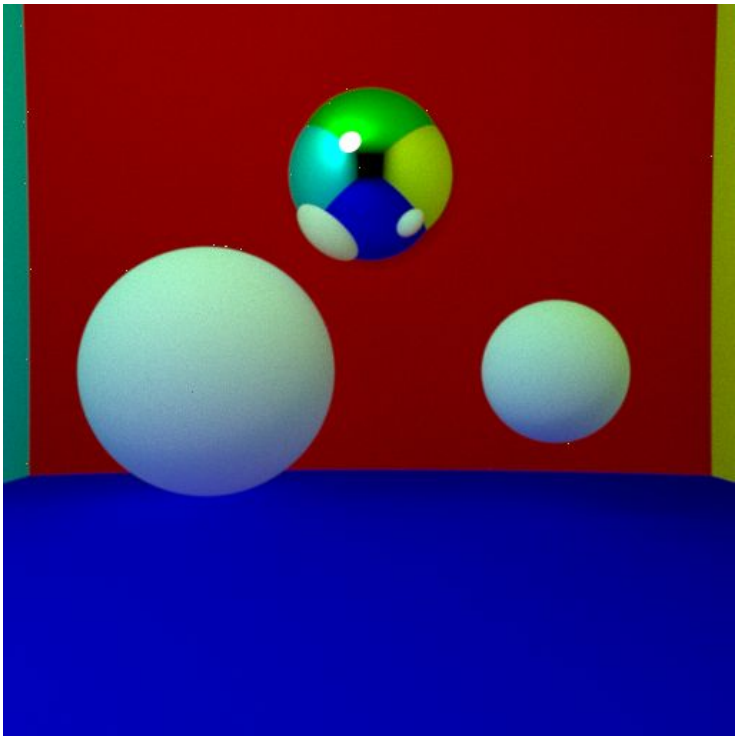

Figure 13 - Set de sphères avec composante de lumière sphérique de rayon 8 - 5000 rayons, grande taille - 20 min d'exécution

IV - Maillages

Pour terminer ce travail, nous avons étudié la construction d'un moteur de rendu avec des scène ne comportant plus des sphères, mais des maillages (sous forme d'ensemble de triangles reliés). Ce code est proposé dans un fichier appelé *main2.cpp* dans la même directory du projet et n'utilise que de la lumière ponctuelle, faute d'avoir un rendu avec des ombres en lumière sphérique.

Pour commencer, la Figure 14 montre que la routine d'intersection d'un triangle est bien implémentée correctement, et que l'on peut, comme avec les sphère, leur choisir une surface type, comme ici, spéculaire.

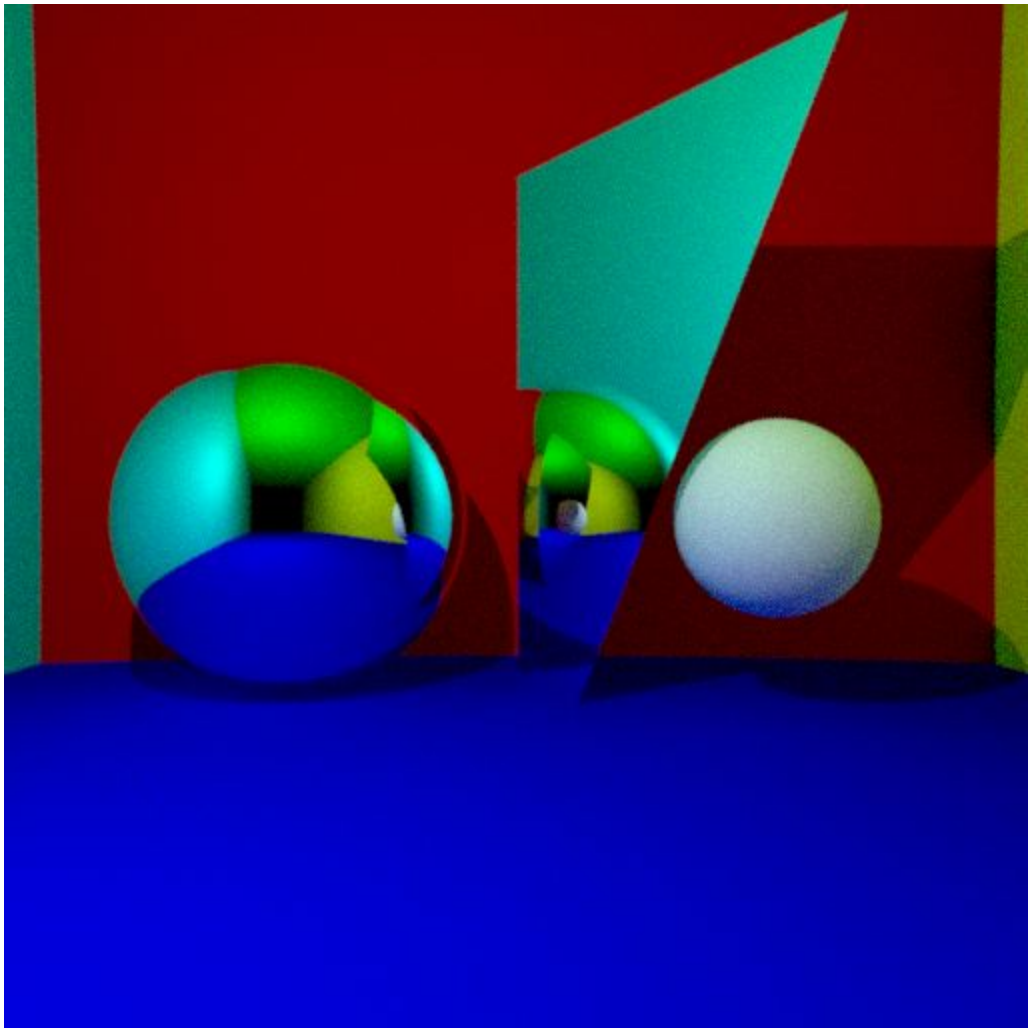

Figure 14 - Set de sphères et triangle miroir en lumière ponctuelle

Puis, on passe à l'utilisation d'un maillage que l'on fait lire au programme. La Figure 15 présente le rendu obtenu en cas de routine d'intersection binaire : s'il y a intersection, le pixel est mis en blanc, sinon, en noir.

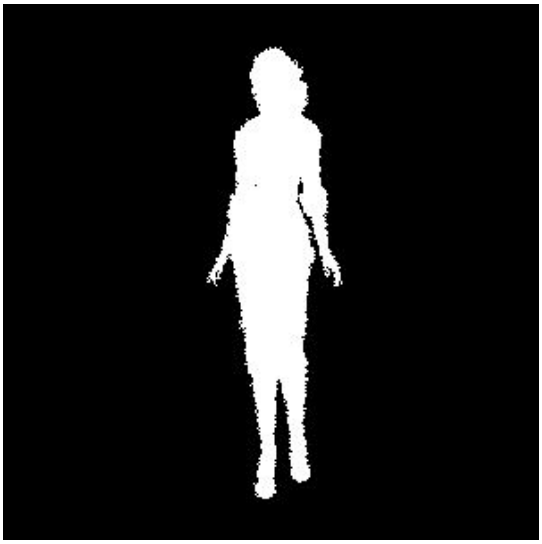

Figure 15 - Maillage de Beautiful Girl en intersection binaire

Lorsque l'on refactore l'ensemble du code pour que les méthode écrites dans le cadre des scènes de sphères soit adapté à la classe Object (dont Sphère hérite maintenant), on voit que les routines d'intersection pour de la lumière ponctuelle fonctionne toujours, et l'on obtient, pour 10 rayons, la Figure 16 où l'on peut distinguer, malgré une qualité bien médiocre, les formes des habits du sujet (pas d'albedo considérée, juste l'inclinaison des triangles face aux rayons incidents).

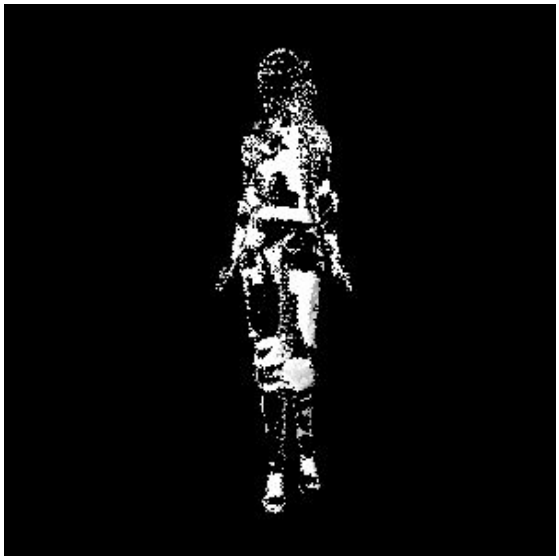

Figure 16 - Maillage de Beautiful Girl avec la routine d'intersection précédente et en lumière ponctuelle

Conclusion

Ainsi, le projet a été formateur, non seulement en raytracing théorique, mais aussi en pratique du C++. En effet, il s'agissait d'un langage qui n'est pas enseigné à l'Ecole Centrale de Lyon dans le tronc commun, et sur lequel j'avais pu suivre un très bref cours il y a plusieurs années. Ceci a entraîné quelques difficultés au démarrage et parfois, sur certains détails syntaxiques, mais, à travers ce projet, j'ai pu comprendre l'intérêt technique et les énormes capacités d'optimisation qu'offre ce langage.

Je ne suis néanmoins pas allé jusqu'au bout du sujet. Ceci est dû, d'abord, à un manque de temps, le cours s'inscrivant dans un semestre relativement chargé ; puis, par difficultés de compréhension, sur l'optimisation du calcul de rendu dans le cas de maillages (avec un nombre très important d'objets à évaluer). Dans la version finale du code rendue avec ce rapport, on pourra voir néanmoins que celui-ci possède une implémentation des objets Bbox et BVHNode décrits en cours. La ligne xxx commentée permet d'enclencher la construction et le tri de ces objets pour la scène en cours de calcul : ce sont des parties de code que j'ai bien implémentées. Néanmoins, je n'ai pas réussi à incorporer les résultats apportés par ces manipulations supplémentaires pour optimiser le calcul d'intersection d'un rayon avec un maillage.

De la même manière, on pourra voir que la constante LUM_SPHERIQUE définie en début de programme et égale à True par défaut permet de passer d'une scène avec lumière sphérique à une scène avec lumière ponctuelle. Enfin, pour les sphère transparentes, dont j'ai commencé l'implémentation, mais, faute de temps, je n'ai pu revenir dessus, et le code est commenté.