Informatique graphique

Thomas Perrot

18 mars 2016

Table des matières

Introduction

Le lancer de rayon est une méthode utilisée pour la synthèse d'image. Elle permet de créer des images très réalistes. Dans le cadre du cours d'informatique graphique proposé à l'Ecole Centrale de Lyon en 2016, nous avons pu expérimenter cette méthode en codant nous-même un programme de lancer de rayon, et en visualisant une scène composée de sphères.

Ce rapport expliquera brièvement l'architecture du code, certains effets mis en place pour renforcer l'aspect autentique des images créées (anti-aliasing, éclairage indirecte, etc.) et quelques ajout personnels (déplacement de la caméra, possibilité de dessiner des formes non-sphériques, etc.).

1 L'architecture du code

1.1 Le langage utilisé

Le langage utilisé pour réaliser ce programme est le Python. Celui-ci permet une grande rapidité de développement, et le code obtenu est très lisible. Ce ne fût pourtant pas un choix optimal pour des raisons de rapidité. En effet, le programme pouvait nécessiter jusqu'à 8h d'exécution pour créer une image sous de la lumière diffuse avec une centaine de rayons par pixels.

1.2 L'architecture globale

Pour maximiser le découplage entre les différents éléments du programme, et faciliter son évolutivité et sa compréhension, le code a été séparé en 3 packages distincts, contenant chacun les classes utiles. Outre ces trois packages, le programme principale main.py va appeler les différentes classes en fonction des actions à effectuer. C'est lui qui pilote le reste du programme, et qui contient les paramètres essentiels. La figure 1 montre l'architecture du programme.

Le package decors Le package décors contient les classes reliées directement à la scène. On y trouve ainsi la classe Camera, la classe Lumière et la classe Scene.

Le package forme Le package formes contient les classes reliées directements aux formes à dessiner sur la scène. On y trouve ainsi la classe Sphere, mais aussi la classe Hyperboloide, et enfin la classe Materiau.

Le package structures Le package structure contient les structures de données nécessaires au programme. On y trouve la classe Intersection, la classe Ray, et la classe Vector.

2 Les matériaux particuliers

2.1 Les matériaux opaques

Pour les matériaux opaque c'est très simple. On projette un rayon et on regarde s'il intersecte une forme de la scène. Si oui, on regarde le produit scalaire entre la normale en ce point et le vecteur lumière arrivant en ce point : l'éclairage de ce point sera proportionnel au max entre 0 et cette valeur. Sinon on retourne du noir. Pour prendre en compte les ombres des objets, on vérifie qu'il n'y a pas d'objet entre le point considéré et la source de lumière. Si on trouve une intersection plus proche que la source de lumière, alors le pixel considéré est dans l'ombre, et on retourne une couleur noire. La figure 2 montre un exemple de scène avec un matériau opaque au centre.

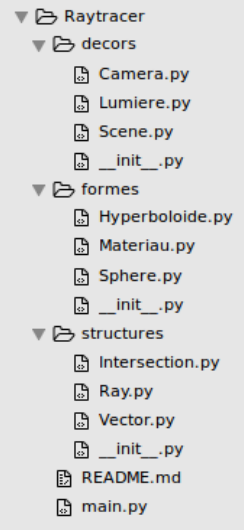

FIGURE 1 – Organisation du code

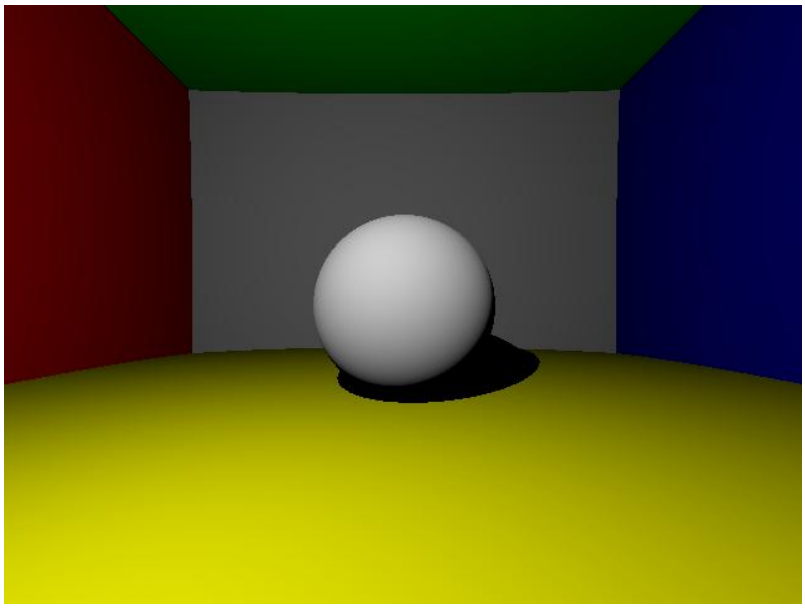

FIGURE 2 – Matériau opaque

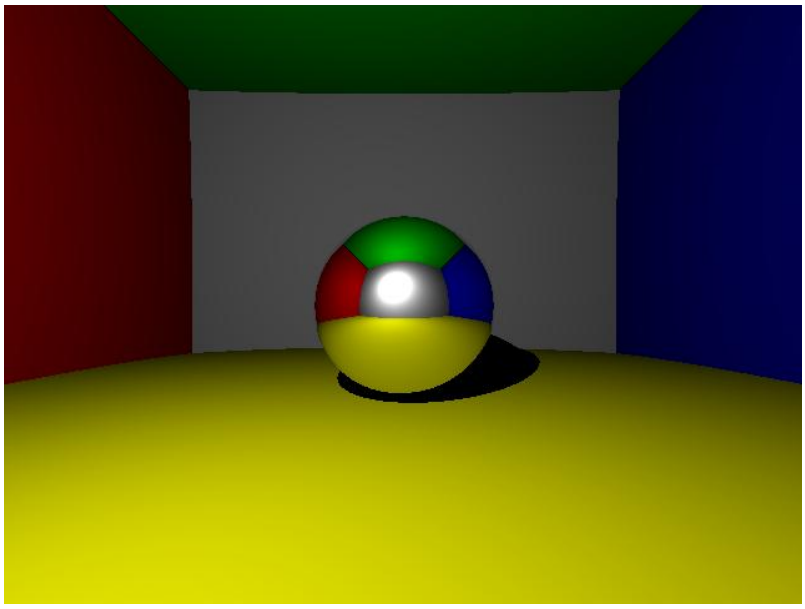

FIGURE 3 – Matériau spéculaire

2.2 Les matériaux spéculaires

Pour créer des matériaux spéculaires (c'est à dire réfléchissants), on teste simplement dans la fonction getColor si le matériau intersecté par le rayon lancé est spéculaire ou non. Si oui, on retourne getColor du rayon refléchi, sinon on retourne la couleur du matériau au point intersecté. La réflexion du rayon est une fonction définie dans la classe Ray qui transforme le rayon en son rayon refléchi au point d'intersection considéré. La figure 3 montre un exemple de scène avec un matériau spéculaire au centre. Pour éviter les rebonds infinis, on introduit un nombre de rebonds maximal (choisi à 5).

2.3 Les matériaux transparents

Prendre en compte les matériaux transparents ressemble beaucoup à considérer les matériaux spéculaires. La différence est qu'on ne considère par forcément le rayon réfléchi, mais souvent le rayon réfracté en suivant les formules de Descartes. La réfraction du rayon est une fonction définie dans la classe Ray qui transforme le rayon en son rayon réfracté au point d'intersection considéré. La figure 4 montre un exemple de scène avec un matériau transparent au centre.

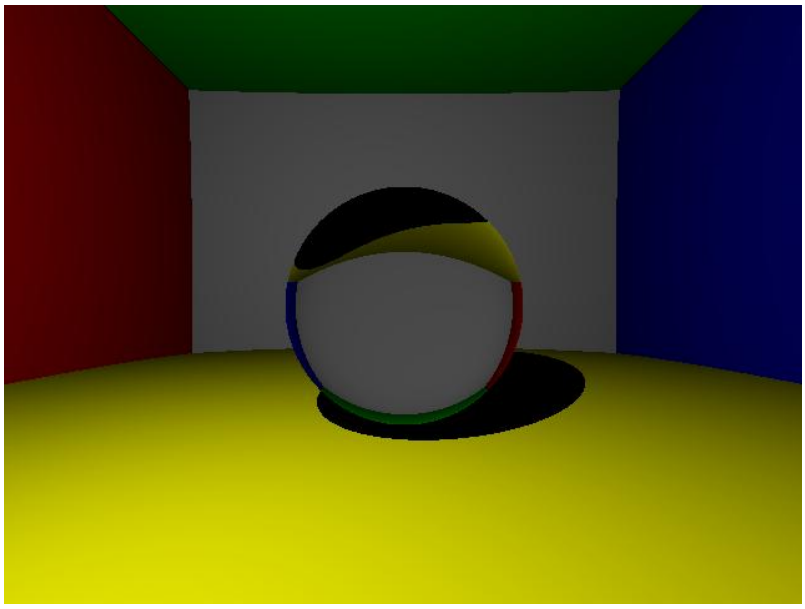

FIGURE 4 – Matériau transparent

3 L'éclairage indirecte

Pourquoi ? L'éclairage indirecte permet de maximiser le réalisme du rendu final. En effet, on s'aperçoit que pour le moment les ombres sont trop marquées. Or, dans la réalité, un élément de surface d'un matériau reçoit de la lumière de toutes les directions qui sont autour de lui, et en réémet à son tour.

Comment ? Pour respecter ce principe, on va utiliser l'équation du rendu pour calculer la couleur de chaque pixel. Cette équation contient cependant une intégrale difficile à calculer. Or, grâce au principe de Monte-Carlo, on sait qu'on peut calculer l'intégrale d'une fonction de manière statistique. Qu'à cela ne tienne : nous allons calculer la valeur de chaque pixel récursivement en "additionnant" les couleurs des matériaux voisins. Pour cela, nous allons projeter plusieurs dizaines de rayons par pixels et à chaque intersection calculer un vecteur refléchi suivant une loi aléatoire. On va ainsi récupérer les couleurs des matériaux voisins du pixels qui sont intersectés par ces vecteurs aléatoires. On obtient alors récursivement la couleur du pixel. Pour de plus amples explications, on pourra se référer au pdf du cours.

Résultat ? Pour 50 rayons par pixels, avec une scène composée de trois sphères au centre (une opaque, une transparente, une spéculaire), on obtient la figure 5. Celle-ci fait aussi appel à l'anti-aliasing (décrit plus bas). Malgré l'utilisation du

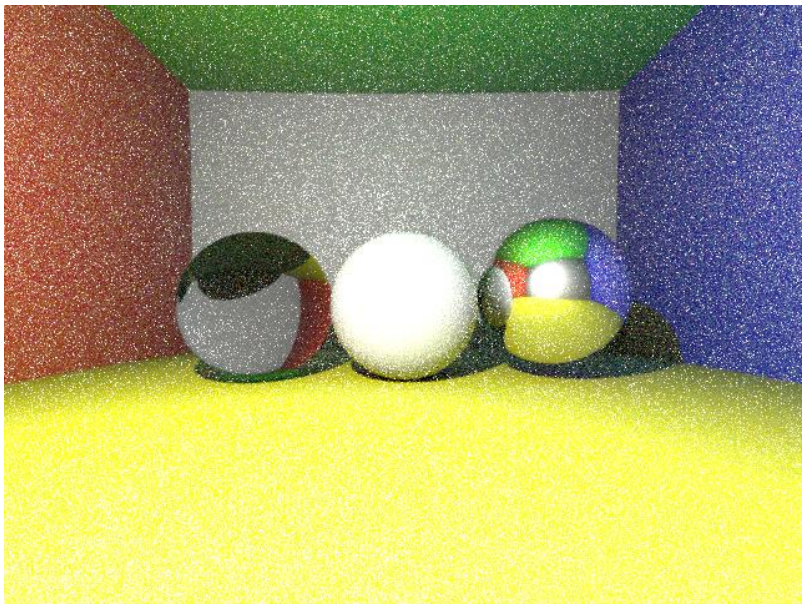

FIGURE 5 – Lumière diffuse et anti-aliasing

parallélisme (détaillé plus loin), et une optimisation de chaque étape du code, on arrive à un temps de calcul d'environ 1h30. Par conséquent, nous ne pourrons essayer avec plus de rayons, même si le rendu actuel manque encore de réalisme.

4 Du flou pour plus de réalisme

4.1 L'anti-aliasing

Le principe de l'anti-aliasing est le suivant : plutôt que de projeter le rayon considéré au centre du pixel concerné, on perturbe légèrement sa direction suivant x et y, et on calcule la couleur correspondante. Pour un seul rayon par pixel, on obtient le résultat visible sur la figure 6. Pour visualiser un anti-aliasing utilisant plusieurs rayons (50), on pourra se référer à la figure 5, et pour un anti-aliasing avec 10 rayons, et un effet de flou lié à l'utilisation d'une distance focale, on se référera à la figure 7. On voit sur ces figures (surtout sur la figure 5) que les limites entre les sphères rouges et grises, entre les sphères vertes et grises, et entre les sphères bleu et grises sont lisses, et non pas crénelées comme on peut le voir très clairement sur les figures 3 et 4 par exemple.

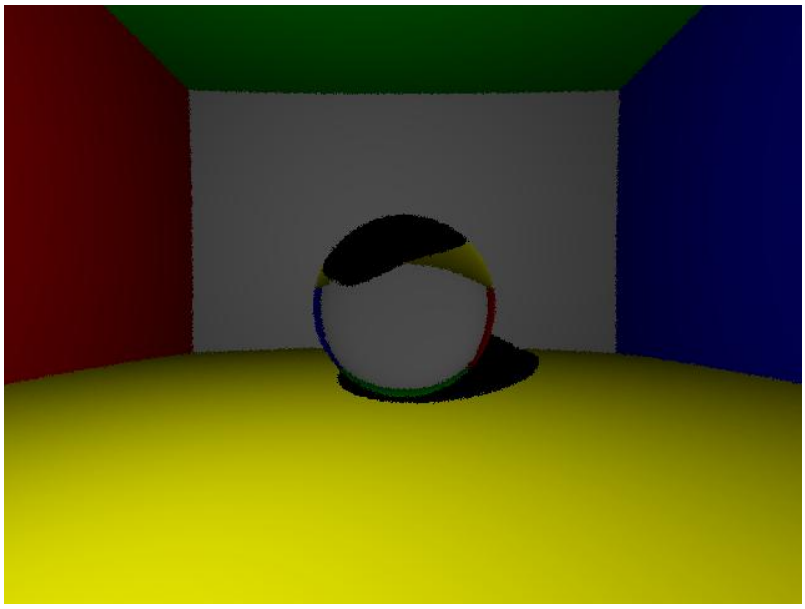

FIGURE 6 – Anti-aliasing pour un seul rayon utilisé

4.2 La distance focale

Pour mettre en place un effet mimant une distance focale, on utilise l'effet suivant. On défini un plan focal dans lequel tous les contours seront nets, et seront de plus en plus flous à mesure qu'on s'éloigne de ce plan. Pour réaliser cela, pour chaque rayon, au lieu de calculer la couleur correspondant à ce rayon, on calcule l'intersection de ce rayon avec le plan focale. On perturbe ensuite légèrement la position de la caméra dans un plan parallèle au plan focal, et on défini un nouveau rayon ayant pour origine le nouvel origine de la caméra, et passant par l'intersection calculée précédemment. Il en résulte un effet de flou sur les pixels, qui s'accroît à mesure qu'on s'éloigne du plan focal. La figure 7 montre un exemple où le plan focal passe par la sphère centrale. L'éclairage est diffus et on utilise l'anti-aliasing, avec 10 échantillons par pixels. On voit très clairement que les contours de la sphère centrale sont nets, mais que les limites entre les sphères du fond de la scène sont floues, par opposition aux limites nettes de la figure 5

5 Le déplacement de la caméra

Pour déplacer la caméra, on crée trois vecteur : direction, up et right. Direction représente la direction dans laquelle la caméra regarde, up le vecteur orthogonal au vecteur direction pointant vers le haut, et right le vecteur pro-

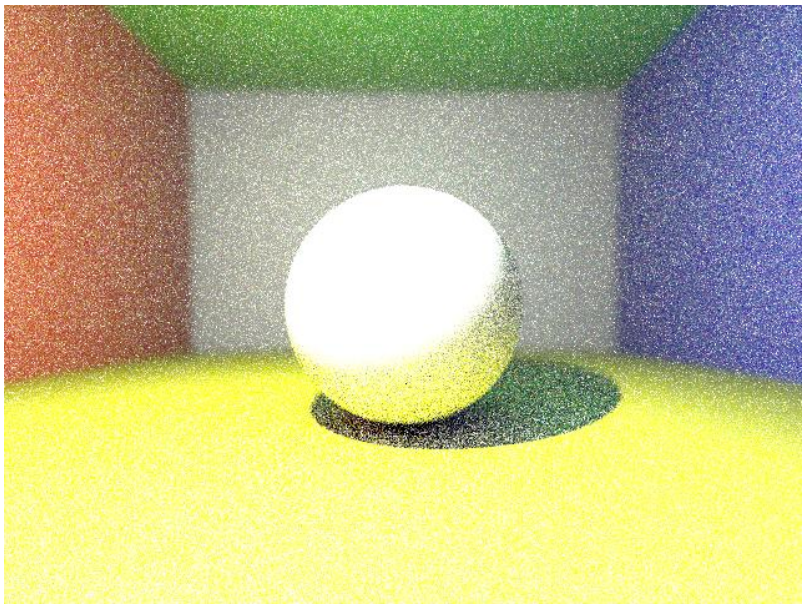

FIGURE 7 – Sphère opaque dans le plan focal

duit vectoriel entre direction et up. On utilise alors chacun de ces vecteurs en multipliant chacune des coordonnées (x, y et z) du rayon par (respectivement) right, up et direction. On obtient après une déplacement de la caméra donnant à la scène une vue plus plongeante la figure 8.

6 D'autres formes

Ceci est un ajout personnel. Pour voir ce que donnerait d'autres formes que des sphères, j'ai choisi de dessiner une hyperboloïde à deux nappes. Le raisonnement est assez similaire à la construction de sphères. On crée une classe hyperboloïde contenant un centre, un coefficient (Vector) contenant les caractéristiques géométriques de l'hyperboloïde (équivalent du rayon pour la sphère), et un matériau. On défini alors une fonction retournant la normale à un point de l'hyperboloïde (obtenue en calculant son gradient) et une fonction déterminant l'intersection entre l'hyperboloïde et un rayon donné (plus compliquée que pour la sphère, mais le principe est le même). La figure 9 montre une hyperboloïde opaque et une hyperboloïde spéculaire.

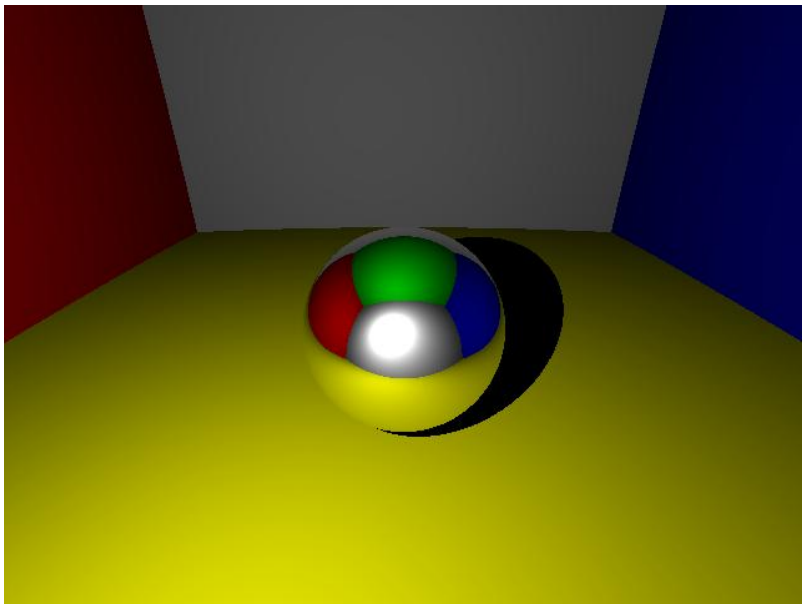

FIGURE 8 – Déplacement de la caméra

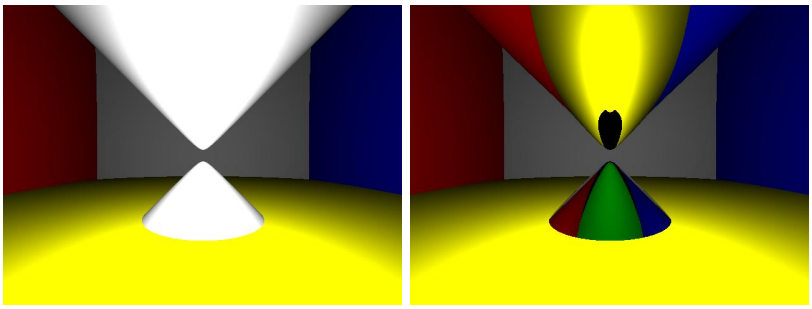

FIGURE 9 – Hyperboloïdes à deux nappes opaques et spéculaires

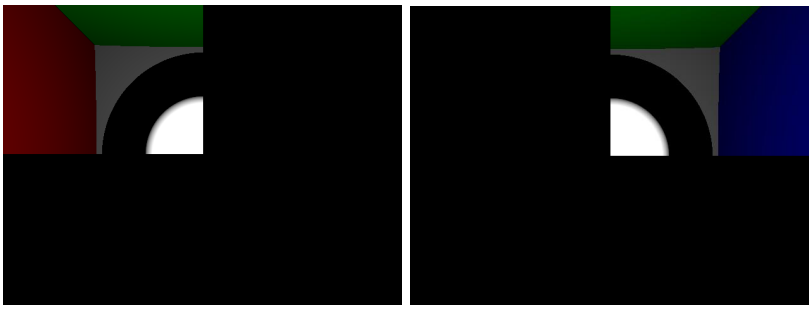

FIGURE 10 – Images issues de processus distincts

7 Le parallélisme

Un temps de calcul élevé Comme précisé plus tôt, choisir Python comme langage de programmation fut fortement limitant en terme de temps de calculs. L'utilisation de Numpy, qui permet d'utiliser des vecteurs déjà définis dans une bibliothèque, n'a pas amélioré les performances (et les a même détérioré, d'où la non-utilisation de cette librairie). De plus, je n'ai pas réussi à utiliser pypy, qui permet dans certains cas d'accélérer le temps de calculs. Pour conclure, le temps gagner à coder en Python plutôt qu'en C++ s'est perdu à attendre la réponse du programme, qui pouvait facilement attendre 2h, même avec du parallélisme.

L'utilisation du parallélisme Pour accélérer le programme, j'ai utilisé la librairie Process, qui permet de faire du multiprocessing. Le travail est alors partagé entre les différents coeurs de mon ordinateur. Comme j'ai 4 coeurs, j'ai utilisés 4 processus en parallèle. Chacun va calculer, indépendamment des autres processus, la couleur des pixels du quart de l'image qui lui incombe. La figure 10 montre une image issue du premier processus, et une image issue du deuxième processus. Une fois chaque processus terminé, on assemble simplement les images en les additionnant. Le gain de temps est significatif : le programme met en moyenne entre 2.5 et 3 fois moins de temps que sans parallélisme. C'est la l'une des grandes forces de la méthode du lancer de rayon : elle s'exécute très bien de manière parallèle, ce qui permet de calculer des images complexes sur des fermes de serveurs.

Conclusion

Nous avons ici présenté quelques techniques pour représenter des rendus réalistes grâce à la méthode du lancer de rayon. Plusieurs constats s'imposent. Premièrement, le choix du langage est primordial. En effet, le programme sous Python peut mettre jusqu'à 8h pour calculer l'image en totalité, ce qui n'est pas raisonnable. Il est donc nécessaire de choisir un langage rapide, qui puisse

d'exécuter rapidement, et si possible se compiler. Deuxièmement, il est très facile de paralléliser la création de telles images. Puisque le calcul de chaque pixel est indépendant du calcul des autres pixels, on peut très facilement demander à plusieurs processeurs de prendre en charge le traitement d'une même image. Cette caractéristique est par exemple utilisée pour générer des images pour des films (par exemple Avatar), en utilisant des fermes de serveurs pour traiter chaque image. Troisièmement et dernièrement, le rendu des images est extrêmement proche de la réalité : les effets de diffusion pour ne citer qu'eux, même s'ils sont issus de phénomènes complexes, sont très bien approximés grâce aux méthodes statistiques.

Pour conclure, la méthode du lancer de rayon fait appel à de nombreux domaines scientifiques (informatique, statistiques, optique, ...), et m'aura permis de voir des utilisations pratiques de disciplines très abstraites, qui trouvent finalement une utilité ici. Plus que dessiner des sphères, ce cours m'a fait comprendre pour de nombreux sujets "à quoi ça sert".

Table des figures