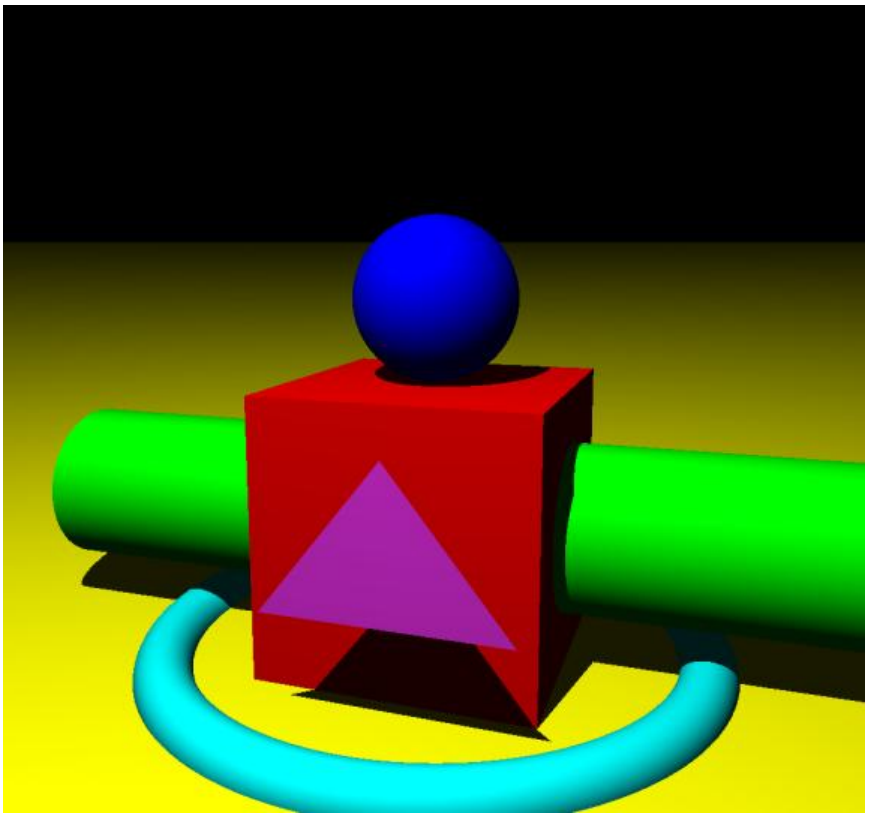
16/01/2011

EISTI

PROJET RAY TRACER



Maël Razavet & Sarah Klelifa

Table des matières

Introduction

Le raytracing est une technique de synthèse d'image simulant le parcours inverse de la lumière. On part de l'objet et on envoie des rayons vers la caméra (représentant l'œil) et vers la ou les lumières présentent dans la scène.
Nous avions donc pour but dans notre projet de réaliser ce principe en langage C.
Notre programme à pour but de lire un fichier contenant une scène écrite selon une syntaxe particulière, et de le retranscrire en une image.
Pour réaliser un tel projet nous avons dû concilier notre connaissance en informatique et en mathématique pour les calculs d'intersections, de normale ainsi que les transformations.

Avancement du projet :

Contenu :

Dans l'archive que l'on vous fournit, il y a le src (code source), le doc (documentation), le bin (l'exécutable), le README, le dossier FichiersPov (contenant nos fichiers .pov), le Doxyfile, le Makefile ainsi que ce rapport.

Structure du fichier :

La lecture de fichier gère les fichiers contenant des commentaires qui sont en dehors des structures définissant les objets, elle gère aussi l'écriture « x, y ou z » pour indiquer l'axe du plan.
Pour les transformations, il faut lui écrire sous forme de vecteur (ex : « scale 2 » devient « scale <2,2,2> »).

Objets traités

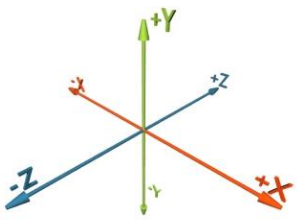
A ce jour, notre programme est capable d'afficher des sphères, des cylindres, des tores, des plans, des cubes, des triangles et des cônes unitaires.
Notre programme gère aussi les transformations scale, rotate, et translate.

Les lumières

Notre programme est aussi capable de gérer plusieurs lumières, et d'afficher les ombres en conséquence.
La lumière à été réalisés avec l'algorithme de Phong, cependant à cause d'une erreur dans la lumière spéculaire celle-ci n'est pas exécuté.
Notre lumière est donc le résultat de la lumière diffuse et de la lumière ambiante sur l'objet.

Notre axe

Nous avons choisit de prendre un axe orienté comme celui ci-dessous :



Ce qui nous resterai à traiter

Avec un peu plus de temps, nous aurions aimé traiter les transformations associer au cône, ainsi que finir toutes les transformations restantes c'est-à-dire union, différence et intersection.
Nous aurions aussi aimé faire une lecture de fichier parfaite, permettant de lire toutes sortes de fichiers, même les plus saugrenus.

Problèmes rencontrés :

1- Pour le binôme Maël :

Je suis plutôt satisfait de ce projet car j'ai pu découvrir beaucoup de choses sur le rendu d'image en informatique, j'ai pu apprendre le langage de programmation C, créer un Makefile complet, générer une documentation très complète, travailler avec un binôme ayant les mêmes exigences que moi (ce qui peut rendre la réalisation du projet plutôt stimulante et peut susciter quelques engueulades)...

Par contre, j'ai trouvé que ce projet était totalement différent des autres que j'ai pu réaliser dans les années précédentes. En effet, j'ai trouvé ce projet beaucoup plus complet et professionnel car on devait utiliser des logiciels comme valgrind, gdb,…. On devait également optimiser notre code le plus possible. Donc le but n'était pas seulement de faire marcher notre projet, il fallait également les outils à notre disposition pour réaliser un projet de manière beaucoup plus professionnelle.
Ensuite, j'ai pu rencontrer quelques difficultés concernant la compréhension du sujet, en effet, il m'a fallu un mois pour vraiment comprendre le principe du raytracing et commencer l'algorithme.
Ensuite, j'ai pu avoir quelques difficultés au début pour apprendre à programmer en C (pointeurs donc problème de mémoire, les segmentations fault,...).

En effet, j'ai dû effectué pendant au moins un mois des recherches sur le raytracer afin de pouvoir créer le planner et de pouvoir expliquer à ma binôme le déroulement de notre projet. En effet, tout notre projet était déjà bien structuré avant de passer à la pratique et donc à la réalisation du projet. Donc, bien que cela peut paraître insignifiant, le fait d'avoir fait de nombreuses recherches, nous a permis de partir sur de très bonne bases et de planifier de manière plutôt correcte les étapes de notre projet.
Ensuite, on avait décidé que je m'occuperais de toute la partie réalisation de l'image de la lecture du fichier à l'écriture du fichier .ppm. Je me suis donc attaqué à cette partie. Puis, le viewplane a été difficile à mettre en place car au début, on n'arrivait pas à orienter l'axe (x,y,z) de manière correcte ce qui nous donné des images différentes de celles de Povray. En plus de cela, Sarah et moi avions pris des axes différents. Il nous a donc fallu pas mal de temps pour rétablir le bonne axe.
Puis, ma tâche suivante était de réaliser toutes les intersections avec les objets demandées (à noter que j'ai également réalisé l'intersection avec un cône mais que celle-ci n'est pas totalement fini). Lors de la réalisation des intersections avec les objets, j'ai eu des difficultés telles que pour réaliser l'intersection avec le tore, il m'a fallu une semaine pour comprendre la méthode de Ferrari et de l'implémenter, puis il a fallu débugger...
Le tore m'a causé beaucoup de problèmes.

J'ai également eu des difficultés concernant les objets infinis (cylindre, cône) pour les avoir en objets finis.
Au niveau de l'intersection avec le cône, elle n'était pas demandée dans le sujet mais je l'ai quand même fait cependant les transformations ne sont pas appliqués sur cet objet.
Pour finir, la lecture de fichier m'a posé problème pour prendre en compte les 3 CSG (union, différence, intersection). En effet, il fallait faire une récursivité pour pouvoir récupérer par exemple, une intersection d'union. Je n'ai donc pas pu finir cette partie de la lecture du fichier à temps et cette version du raytracer ne permet pas complètement de traiter toutes les opérations sur les 3 CSG.

Cependant, vu notre algorithme, il est possible de faire l'intersection entre deux objets et seulement deux objets. La différence est possible mais elle est peut-être incorrect (on n'a pas testé tous les cas) entre deux objets (uniquement deux objets différents). Au niveau de l'union, on sait que l'union entre deux objets est l'image normale du raytracing mais ce qui est intéressant c'est de pouvoir faire une intersection d'union ou un translate sur toute une union par exemple. On n'a pas eu le temps de prendre en compte tout ceci.

Pour finir, je trouve que notre binôme a plutôt bien fonctionné car comme je l'ai di précédemment, nous avions les mêmes exigences et notre but était de faire un projet complet et bien fait. Je pense que cet objectif était plutôt réussi mais ce n'est que mon avis. Si c'était à refaire, je me remettrai avec Sarah.

2- Pour le binôme Sarah :

Il nous a été assez difficile de bien s'investir dans le projet au début car on le voyait comme quelque chose d'assez ambitieux pour notre niveau. On ne savait pas trop comment partir et on ne savait pas vraiment ce qu'on attendait de nous, même en lisant l'énoncé du projet à plusieurs reprises.
Une fois le but du projet compris, nous n'arrivions pas vraiment à visualiser la manière de procéder. On savait qu'il fallait lancer des rayons sur des objets qui avaient une équation mathématique pour ainsi les retranscrire à l'écran.
Lancer des rayons d'accord mais comment?
Est ce qu'il faut créer un programme qui dessine des rayons et qui regarde l'intersection avec l'objet?
Plein de questionnement nous venait à l'esprit sur la façon de procéder.
On décida donc de s'attaquer à la chose la plus évidente : la lecture de fichier, qui paraissait assez simple.

Ah! Mais j'avais oublié, c'est en C qu'on doit coder le raytracer!
En C, langage encore jamais étudié, après les quelques cours en classe, ce langage commencer à devenir plus compréhensible jusqu'à que l'on s'attaque au pointeur...
En effet les pointeurs, pas vraiment facile à comprendre, au début on croit avoir compris, puis on s'attaque au TP des listes et on se rend compte que les histoires de pointeur ce n'est pas si facile que ça.
En effet il a fallu réussir à comprendre qu'on été en train de manipuler des adresses, que cela nous facilitait la vie d'un coté car il suffisait de mettre l'adresse de la variable à modifier en paramètre d'une fonction pour nous éviter de faire une fonction qui renvoi quelque chose mais une procédure, en effet la variable se modifier toute seule!
Cependant combien de fois on oubliait que lorsque l'on mettait une variable en paramètre de fonction, la valeur de celle - ci était copiée pour la fonction mais malheureusement jamais modifié, on se retrouvait avec des résultats aberrant alors que le seul problème était qu'il fallait mettre l'adresse de la variable.
Et allait chercher l'erreur quand ceci n'est pas instantané chez vous…
A part l'histoire des pointeurs, on a eu du mal avec les headers, en effet une fois avoir compris comment ça marchait et à quoi ça servait, nous avons était confronté au problème des inclusions infinies.

En effet il nous arrivait d'avoir besoin d'un fichier.c dans notre code, donc on l'incluait, cependant dans l'autre code le fichier été déjà inclue donc problème!
Cela nous donnait souvent une inclusion infinie qu'il fallait résoudre, à l'époque il été assez difficile de comprendre comment ça marcher.
Au final nous avons d'ailleurs décidé de mettre des "ifndef" dans chacun des fichiers pour être sûre que les problèmes de compilation ne venaient pas de là.

Après avoir partiellement compris les pointeurs (partiellement, car je suis sûre que les pointeurs nous cachent encore de grandes spécialités inexploités), et les ifndef des fichiers.h parlons un peu de cette lecture de fichier.
Il est vrai que il nous a était plusieurs fois dit qu'il fallait mieux faire les algorithmes sur papier car il était moins dure de jeter une feuille d'algorithme entière qu'un fichier entier de code.
En effet voulant tout de suite commencer à coder, j'ai été confronté au dilemme du jetage de fichier.
Il est vrai que la lecture de fichier n'est pas à proprement dit quelque chose de très compliqué.
Elle devient compliquer lorsqu'on veut la coder en C..
En effet, je pense que le C n'est pas le langage le plus adapté pour gérer des chaines de caractère, du coup il m'a fallut étudier tout le fonctionnement des tableaux de caractère qui n'est pas quelque chose de très facile à comprendre au début.
Après plusieurs essaie de code, sans algorithme, directement sur l'ordinateur, qui étaient la plus part du temps inefficaces, je me suis donc résolue à écouter les conseils donnés.
Je pris un papier et un crayon et commença à créer mon algorithme pour la lecture de fichier, quel changement!
Tout devenait plus clair, puis si on se trompe, on efface et on recommence, tout ça sans un énorme pincement au cœur!
Je décidai donc de réaliser tout mes algorithmes sur papier, sage décision, non?
Le temps que je faisais la lecture de fichier Maël cherchait des informations sur la façon dont on pouvait s'y prendre pour coder ce raytracer.

Une fois qu'il eu obtenue un nombre suffisant d'information, que les exposés étaient passés, et que la lecture de fichier était "valable" nous avons décidé de se mettre à coder le rendu d'image.
Il est vrai que coder une lecture de fichier et voir que cela lit bien un fichier ne procure pas un plaisir fou, alors que voir une image affichait à l'écran, relève de l'exaltation intense!
Bon l'image n'est pas apparut de suite, il a fallut tout d'abord que Maël code l'intersection entre une sphère et un rayon après avoir enfin compris qu'un rayon c'était juste l'équation d'une droite.

Il nous a aussi fallu comprendre le fonctionnement d'un fichier ppm, il faut avouer que le fichier ppm ne nous a pas posé de problème particulier.

Lorsque Maël me passa un merveilleux coup de fil pour me dire qu'il réussissait à afficher une sphère, toutes mes craintes c'était à moitié envolé.
Ce n'était pourtant qu'une simple sphère 2D, mais rien que de voir qu'on avait déjà réussit à obtenir quelque chose me remplissait de joie.

Ensuite, une des choses qui nous a posé le plus de problème était le view plane.
Sacré viewplane, d'abord pour se représenter ce qu'est un view plane, il m'a fallu personnellement plus de deux mois, puis réussir à le faire marcher à peu prés 2mois et demi...

Le problème qu'on a eu c'était que s'étant partager les taches chacun réaliser son code de son coté et puis on mettait tout en commun.
Le problème c'est que Maël et moi n'avions pas pris le même repère, du coup l'image été toute désorienté.

Après de multiple essaie pour comprendre d'où venait l'erreur nous nous sommes dit qu'on verra ça plus tard et qu'on allait quand même avancer.

Je décidai donc de me mettre à coder les lumières, ayant faite quelque rechercher, cela me paraissait pas très compliquer, enfin ça ne ma tout de même pas empêché de passé tout le weekend dessus.
En effet comme on avait des problèmes de viewplane la couleur s'affichait assez mal. N'ayant pas fait le rapprochement entre le viewplane je cherchais qu'est ce qu'il pouvait bien clocher dans ma formule.

Après avoir réussit à trouver, ce qui ne marchait pas, et l'avoir résolu, je m'attaquais aux transformations car nous avons décidé de faire des intersections unitaires.

Pour cela il a fallu recoder toute les fonctions sur les matrices, certaines étaient assez faciles, mais la fonction qui inverse la matrice m'a énormément posé de problème.
En effet il n'était pas simple de se visualiser comment procéder, car je ne voyais pas comment coder une fonction qui regarder toute seule ou été la ligne qui nous arranger et qui permettait de faire le pivot.
Finalement j'optais pour une méthode beaucoup moins optimisé qui était celle du calcul du déterminant.

Ne faisant pas attention à l'optimisation de mon code, je calculais pour chaque point l'inverse de la matrice de l'objet, alors que l'inverse de la matrice de l'objet ne change pas.
Je ne comprenais donc pas pourquoi mon programme mettait plus de 6mn à s'exécuter.

Après de longue recherche, j'eu compris qu'il fallait simplement initialiser la matrice inverse de la transformation au début, dans l'initialisation des structures des objets et que cela nous évitait énormément de calcule.
Il a fallut aussi comprendre qu'il fallait libérer la mémoire avec les variables dont on ne se servait plus car sinon le programme mettait du temps à s'exécuter car la mémoire se vidait et ça allait dans le swap.

Une fois avoir tout optimisé, je m'attaquai à la perfection du cylindre, en effet nous n'arrivions pas a calculer l'intersection avec les extrémités.

Puis la lumière ne s'affichait pas bien, je cherchais donc le problème au niveau de la normale, en me rendant compte à un jour du rendu qu'il suffisait seulement de normaliser les normales.

Ce que ça nous a apportés :

Ce projet m'a énormément appris, à me débrouiller par moi-même, à faire des recherches quand quelques chose ne me paraissait pas très claire.
Il m'a aussi appris le langage C, car nous sommes partit d'aucune connaissance en ce langage pour réussir à réaliser un tel projet.
J'ai également appris à ne pas baisser les bras même quand quelques parait trop ambitieux car le plaisir d'y être arrivé est vraiment quelque chose de merveilleux.
Je suis assez content d'avoir eu un binôme aussi motivé que moi, nous nous sommes équitablement répartit le travail ce qui fait que nous avons relativement bien avancé.
Je suis assez fière de voir ou nous sommes arrivés, et malgré des moments de grands stresse, d'énervement, de manque de confiance en soi, je peux dire que j'ai pris un grand plaisir à coder ce projet.