PROJET DE MODELISATION GEOMETRIQUE

Projet réalisé à l'esil en deuxième année du département Informatique

Groupe de projet : ThierryBaud, Djemel Guizani, Franck Rinaudo

Introduction

Le but de ce projet était de créer une interface graphique utilisant OpenGl et un environnement utilisateurs GLUI, pour développer la structure d'un raytracer. Le projet devait comporter des éléments de géométrie analytique tel que le calcul d'intersection entre une droite et une sphère, entre une droite et une boite ou entre une droite et un cylindre.

L'équipe de travail

Notre groupe est composé de trois personnes : Thierry Baud, Djemel Guizani et Franck Rinaudo. Pour nous contacter vous pouvez envoyer un mail à Dguizani@esil.univ-mrs.fr, Frinaudo@esil.univ-mrs.fr, TBaud@esil.univ-mrs.fr.

Chapitre 1 : Description de la structure du programme

Un rendu en raytracing consiste à évaluer l'intersection d'un rayon (une ligne droite) avec les objets de la scène. Commençons par le plus simple : on ne considère pas les rayons réfléchis et transmis (autrement appelés rayons secondaires), nous ne gardons que les rayons primaires.

L'algo pour tracer un seul rayon est le suivant :

- Pour un point (x,y) de l'image voulue générer le rayon : point d'origine et direction
- Faire l'intersection du rayon avec la scène (parcourir tous les objets et rechercher les intersections)
- Récupérer les coordonnés des intersections.
- Pour aller plus loin on pourrait en suite récupérer la couleur de l'objet intersecté en premier.

Pour décrire une scène nous avons eux besoin de créer une structure de données qui permettait de simplifier le traitement des objets. Nous avons développé chaque objet de la scène comme une classe C++ qui hérite d'une classe générique (Objet3D) c'est à dire que toute action se fera de la même manière que l'on veuille traiter une sphère ou une boite.

Cette spécificité, nous permet de traiter une liste d'*Objets3D* auxquelles on applique des méthodes et non plus des objets indépendants auxquels on applique des fonctions différentes. Traiter tous les objets revient donc à parcourir la liste et à appliquer une méthode sur chaque élément.

Description des classes développées dans le programme :

Point : une classe simple qui permet de stocker les coordonnées d'un point en 3D et de faire quelques manipulations sur ce point.

Vecteur : Une classe simple aussi qui permet de faire des manipulations générales sur un vecteur en dimension 3.

Objet_3D : C'est la classe de base pour définir tout objet ajouté au projet. Elle permet de créer une liste d'objet 3D qui sera parcouru lors de la detection d'intersection.Cette classe ne defini que des méthodes virtuelles, tel que WireDraw, SolidDraw, AjouterDesControlesGLUI, qui permet de simplifier le code principal (ie : Pour dessiner tous les objets on parcours la liste et on appelle la fonction SolidDraw pour chaque élément. C'est le compilateur qui choisira quelles fonctions de dessin il faut appeler). (cf. Figure 1)

Liste : Une classe que nous avions déjà développé pour un autre projet et qui nous a permis de gérer les listes d'objets, les listes de rayons et les listes d'intersections très rapidement.

Rayon : Cette classe stocke les informations sur un rayon lumineux. Pour l'instant on elle ne comporte que les coordonnées du point d'origine et du vecteur directeur dans le repère global.

Elle pourrait de plus servir a connaître la couleur du rayon et son intensité si ce rayon était émis par une lumière.

Intersection : Cette classe comporte toutes les données relatives à une intersection entre un rayons et un objet3D : Elle permet de savoir quel est l'objet intersecté, le point d'intersection en coordonnées globales.

Pour la rendre réelement utilisable dans un vrai raytracer il faudrait lui rajouter : la normale à l'objet au point d'intersection, car cette normale permet de calculer le rayons transmis et le rayon refléchi selon les propriétés de l'objet 3D.

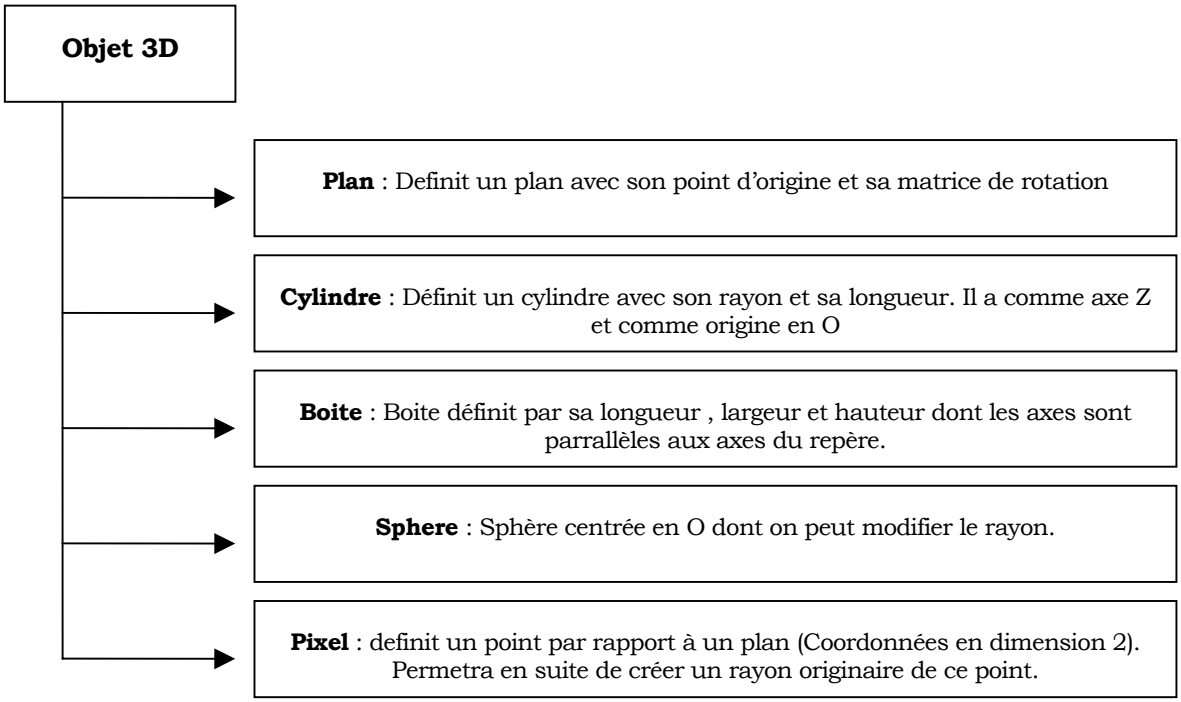

Figure 1

Chapitre 2 : Ce que nous avons géré

Le but était plus de montrer les mécanismes d'un raytracer que de programmer un raytracer. Seul la structure de base d'un raytracer a été développée, aucun traitement de lumières n'a été implanté. Nous avons surtout développé une interface graphique qui permette de vérifier visuellement si un rayon émit est en intersection avec un objet.

Nous avons aussi développé des méthodes d'intersection pour chaque objet 3D (Sphère, Boite, Cylindre), qui prennent un rayon en paramètre et qui rendent une liste d'intersections (ie : La coordonnée de l'intersection et le nom de l'objet).

2.1. Gestion des intersections :

2.1.1. Intersection Rayon-Sphère

Cette intersection est relativement simple et rapide à obtenir. Elle se base sur la distance d'un point a une droite : si le centre de la sphère est à une distance supérieur au rayon : il n'y a pas d'intersection. sinon il y a une ou deux interection(s) (ie : soit le rayon est tangent soit il coupe la sphère).

Algorithme :

- Soit A le projeté de O le centre de la sphère sur le rayon.
- On calcule la distance entre A et 0.
  - Si dist > Rayon_Sphere alors il n'y a pas d'intersections.
  - Si dist = Rayon_Sphere alors il y a une intersection en A.
  - Sinon dist < Rayon_Sphere et alors on calcule la demi corde = racine (R^2-$dist^2$) et on obtient les 2 points A1 = A + c.V et A2 = A – c.V ou c est la demi-corde et V le vecteur directeur normé du rayon. (cf. figure 2)

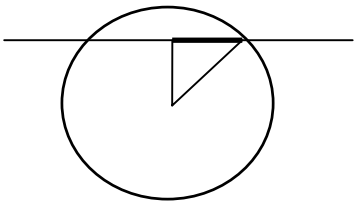

Figure 2

Pour les deux autres intersections nous avons cherché à résoudre les équations paramétrique fournit lorsque qu'un point appartient en même temps à la droite et à l'objet.

Nous sommes partie de la définition analytique d'un rayon : Soit (x1, y1, z1) le point d'origine du rayon et la direction du rayon (i, j, k)

La paramétrisation du rayon donne,

- X = x1 + i * t
- Y = y1 + j * t
- Z = z1 + k * t

t représente donc la distance sur le rayon entre les points (x1,y1,z1) et (X,Y,Z), ce qui peut permettre de sélectionner l'intersection la plus proche (pour enlever les faces cachées, et calculer la couleur du pixel en fonction du premier objet intersecté : ceci représente le premier pas d'un « vrai » raytracer).

2.1.2. Intersection Rayon-Boite

Pour cette intersection nous avons eu besoin de changer de repère : Nous devons nous placer dans le repère de la boite. Puis on fait des intersections Droite-Plan et on vérifie que le point d'intersection est bien inclu dans la face du cube. On applique cette technique pour les six faces.

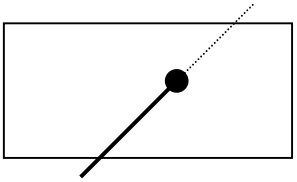

Figure 3

Intersection droite-plan : nous ne considérons que des plans orthogonaux à l'un des axes du repère (On est dans le repère de la boite).

Exemple : pour le plan orthogonal à Z. On fixe Z pour les coordonnées du point sur le rayon et on teste si le (X,Y) est bien a l'intérieur du rectangle.

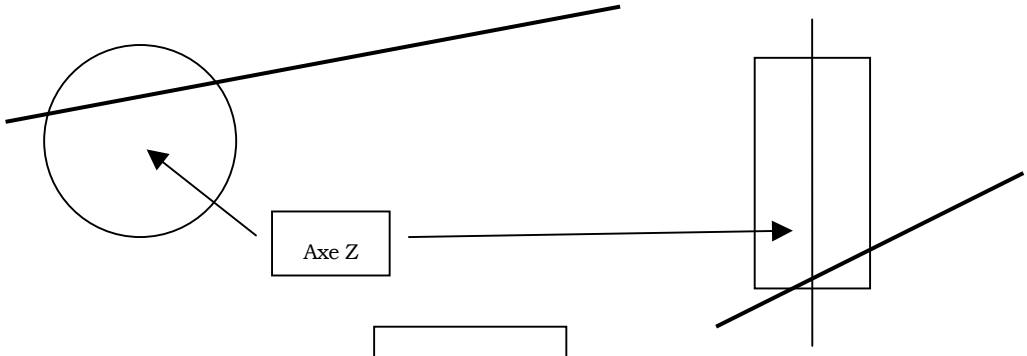

Figure 4

2.1.3. Intersection Rayon-Cylindre

Pour cette intersection aussi nous avons eu besoin de nous placer dans le repère du cylindre. Une fois cela fait il suffit de vérifier si :

- Le rayon intersecte le cercle dans le plan XY. Ici il faut faire une intersection Doite–Cercle en 2D. Nous avons choisi de le faire de facon analytique en résolvant l'équations du cercle : $||OA||^2=R^2$.
- Les coordonnées du point d'intersection sont bien comprises entre 0 et la hauteur du cylindre sur l'axe Z.

Conclusion et évolutions possibles

Pour continuer de développer un raytracer Il faudrait d'abord terminer la génération d'une image (un bitmap) qui rendrait compte des intersections avec des couleurs différentes selon l'objet intersecté. Puis on devrait rajouter le traitement de sources lumineuses ce qui est une grosse partie car elle nécessite la gestion des propriétés des objets (matières, textures…) mais aussi il faudrait aussi calculer les rayons transmis et réfractés et appliquer une récurrence sur le lancer de rayon.

C'est un Projet très interressant qui serait à la fois amusant et utile de poursuivre. Il nous a permis de nous familiariser avec des outils très performants tels que la librairie OpenGl et toutes ces librairies auxiliaires.