Infographie

[INFO0026-3]

Professeur E. Béchet

Rapport gray :: ajout d'objets géométriques

Alewaerts Frédéric
Elias Jefferson

1ère Master en Sciences Informatiques

Année Académique 2009 - 2010

Table des matières

1 Introduction

Ce présent document constitue notre rapport pour le projet à réaliser pour le cours d'infographie dispensé par M. Béchet (Université de Liège). Chaque groupe s'est vu attribué un sujet. Tous les sujets consistaient soit à améliorer, soit à ajouter des fonctionalités à un programme déjà existant appelé `gray`. Notre partie du travail consistait en l'ajout de formes géométriques, car il n'existait alors que des sphères.

Quelques modifications dans l'agencement ont été apportées pour permettre un découpage entre les fichiers source et les fichiers exécutables ou les fichiers générés par l'utilitaire `CMake`. En effet, vous trouverez sur le dépôt svn un dossier `src/` qui contiendra les fichiers sources et l'indispensable `CMakeList.txt`. Vous trouverez aussi un dossier `build/` qui va servir de répertoire où générer les fichiers. La manière de faire est la suivante :

```
gray$ cd build
build$ cmake ../src
build$ make
```

Cet ensemble de commande vous permettra de construire le fichier exécutable. Cependant, notre phase nous a permis de nous rendre compte que le projet de cette année était fortement orienté vers des opérations sur des sphères uniquement, ce qui fait que nos structures sont inutilisables dans la version finale de `gray`. Nous avons donc créé un dossier `job2` dans lequel nous avons repris une des premières version du travail et y avons inclus nos modifications. Nous avons d'abord pu observer que l'application de la couleur ne se fait pas comme cela devrait dans la version finale de `src/` : aucune cohérence, alors qu'avec la version de départ, les couleurs sont correctement assignées. Nous vous encourageons donc à effectuer vos tests avec les fichiers du dossier `job2`.

Remarque : M. Bechet nous avait suggéré de créer une interface entre sa librairie `gnurbs` et notre programme, et de l'utiliser pour ajouter, par exemple, une surface de Bézier. Le fait est que nous n'avons pas eu l'accès nécessaire pour utiliser cette librairie, et les examens approchant, nous préférons nous concentrer dessus plutôt qu'attendre. Néanmoins, nous tenons à faire remarquer qu'en utilisant les différents objets 2D que nous avons implémentés, il serait aisé d'implémenter n'importe quel type de surface.

2 Formes géométriques planes

Dans cette section, nous allons présenter les 3 types de formes géométriques planes que nous avons ajoutées à la librairie, à savoir :

> un triangle ;
> un parallélogramme.
> un polygone convexe

Dans les sous-sections qui vont suivre, nous ne présenterons que les éléments réellement importants de l'implémentation, c'est-à-dire la méthode utilisée pour renvoyer un vecteur normal au plan (même si cela s'avère relativement trivial) ainsi que les méthodes d'intersection.

2.1 Vecteur normal au plan

Vu que nous disponsons d'une liste de côtés, il est aisé d'en extraire deux vecteurs directeurs, soient v_1 et v_2. Le vecteur normal au plan décrit par la figure géométrique est le résultat du produit vectoriel de ces deux vecteurs, soit :

$$N = v_1 \times v_2 \tag{1}$$

2.2 Intersection entre un rayon et un triangle

L'algorithme que nous avons implémenté est basé sur [3].

Un rayon lumineux ne fait jamais que définir une demi-droite. Une droite peut être définie paramétriquement ou vectoriellement. Dans le second cas, on aura une équation du type :

$$R(t) = O + tD \tag{2}$$

où R est le nom de la droite (ici, celle définie par le rayon), O représente l'origine du rayon, i.e. la position de la source lumineuse et D représente le vecteur directeur de R i.e. la direction empruntée par le rayon lumineux.

Soit $R(t)$ un rayon lumineux, soit P_1,P_2,P_3 trois points de l'espace non alignés, qui définissent donc un triangle. Nous allons chercher l'intersection entre $R(t)$ et ce triangle.

Si nous retournons dans le cours de géométrie de M. Lecomte, nous pouvons y voir définies les coordonnées barycentriques. Nous allons les utiliser pour définir le fait suivant :

Tout point $T(u, v)$ est intérieur au triange $P_1P_2P_3$ si et seulement si

$$T(u, v) = (1 - u - v)P_1 + uP_2 + vP_3 \tag{3}$$

où (u, v) sont les coordonnées barycentriques de T et doivent respecter les conditions suivantes :
- $u, v \geq 0$
- $u + v \leq 1$

Nous cherchons à déterminer s'il existe une intersection entre $R(t)$ et le triangle. Cela se traduit par la recherche de l'existence d'un point T tel que :

$$R(t) = T(u, v) \tag{4}$$

Ce qu'on peut réécrire

$$O + tD = (1 - u - v)P_1 + uP_2 + vP_3 \tag{5}$$

Ou encore sous forme matricielle par bloc :

$$[-D, P_2 - P_1, P_3 - P_1] \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - P_1 \tag{6}$$

Posons $V_1 = P_2 - P_1$, $V_2 = P_3 - P_1$ et $T = O - P_1$. La solution à l'équation (6) est obtenue par la résolution d'un système

$$A_{3x3}\ x_{3x1} = b_{3x1}$$

par l'utilisation de la méthode de Cramer par exemple qui consiste en l'assignation suivante :

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{\det A} \begin{bmatrix} \det([T, V_1, V_2]) \\ \det([-D, T, V_2]) \\ \det([-D, V_1, T]) \end{bmatrix} \tag{7}$$

Rappel d'algèbre linéaire

$$\det([A, B, C]) = -(A \times C) \bullet B = -(C \times B) \bullet A \tag{8}$$

On peut donc réécrire l'équation (7) comme suit :

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{(D \times V_2) \bullet V_1} \begin{bmatrix} (T \times V_1) \bullet V_2 \\ (D \times V_2) \bullet T \\ (T \times V_1) \bullet D \end{bmatrix} = \frac{1}{P \bullet V_1} \begin{bmatrix} Q \bullet V_2 \\ P \bullet T \\ Q \bullet D \end{bmatrix} \tag{9}$$

en posant $P = (D \times V_2)$ et $Q = (T \times V_1)$

2.3 Intersection avec un parallélogramme

La méthode est relativement similaire à celle du triangle : les concepts utilisés sont les mêmes, à savoir la détermination des coordonnées barycentriques u et v.

Tout point $T(u, v)$ est intérieur au parallélogramme $P_1P_2P_3P_4$ si et seulement si

$$T(u, v) = (1 - u - v)P_1 + uP_2 + vP_3 \tag{10}$$

où (u, v) sont les coordonnées barycentriques de T et doivent respecter les conditions suivantes :
- $u, v \geq 0$
- $u \leq 1$ et $v \leq 1$

En particulier, on a :

$$P_4 = P_2 + P_3 \tag{11}$$

La seule différence avec l'algorithme appliqué au triangle sera donc la comparaison des valeurs de u et v.

2.4 Intersection avec un polygone convexe

Vu que nous avons cherché non pas à avoir un polygone spécifique mais bien un polygone convexe quelconque, il nous a donc fallu trouver un moyen de rester général pour la méthode d'intersection. En fait, elle requière un traitement préalable qui est fait lors de l'instanciation.

Ce traitement préalable consiste à découper le polygone en un ensemble de triangles. Nous discuterons de la méthode ci-dessous. L'intersection revient alors à tester l'intersection du rayon avec tous les triangles qui auront été créés à l'instanciation.

Pour ce qui est de l'algorithme utilisé pour le découpage en triangle, il est purement sorti de notre imagination et consiste à utiliser le *design pattern* appelé **Divide and conqueer** [1] qui consiste à réduire un problème complexe en un ensemble de problèmes plus simples en utilisant la récursion à bon escient.

> Soit N le nombre de sommets qui composent le polygone. Dans la structure, on commence à compter les sommets à 0. Cependant, ce sommet constituera le point commun entre tous les triangles. Nous appellerons donc une procédure récursive avec comme paramètres initiaux $(1, N-1)$. Dans la suite, le paramètre de gauche sera appelé `start` et le paramètre de droite `end`.
>
> 1. Calculer `diff = end - start`.
> 2. Si `diff` vaut 1, nous avons un triangle constitué des sommets 0, `start` et `end`.
> 3. Sinon, calculer le "pas" comme étant la division entière de `diff` par 2 : `step = diff / 2`
> 4. Appeler la procédure récursive avec comme paramètres (`start` , `start+step`)
> 5. Appeler la procédure récursive avec comme paramètres (`start+step`,`end`)

Une autre technique aurait pu être implémentée : vu que le polygône est convexe, son centre de gravité se trouve forcément à l'intérieur du polygone. Il aurait suffit de relier ce point avec chaque côté du polygone, nous avons préféré jouer le coup de l'originalité.

3 Formes géométriques

Dans cette section, nous allons vous présenter les 2 formes géométriques que nous avons ajoutées à la librairie, à savoir :

1. *G6faces* : une figure géométrique convexe à 6 faces et donc ayant pour bases des quadrilatères convexes étant au moins des parallélogrammes.
2. *Gcylinder* : un cylindre de révolution.

Dans les sous-sections qui vont suivre, nous allons présenter les détails non triviaux de l'implémentation (Représentation par facettisation pour le cylindre, calcul de la normale en un point et pour finir l'intersection).

3.1 Représentation d'un `G6faces`

Pour représenter un objet `G6faces`, nous utilisons simplement des objets `Gparallelog`. Ainsi, dans notre implémentation, la classe `G6faces` maintient une liste de 6 `Gparallelog`,

représentant les 6 faces du `G6faces`. Au passage, nous ajustons la définition de chaque face afin que la normale soit orientée vers l'extérieur.

3.2 Normale à un `G6faces`

La normale est maintenant renvoyée lors de l'appel à la méthode qui demande la première intersection. Cependant, nous avons laissé accessible la méthode pour récupérer la normale. Vu qu'en règle générale, on la demande seulement lorsqu'on a confirmé qu'il y avait intersection, et vu qu'une instance de `G6faces` utilise 6 `Gparallelog` en tant que représentation, il suffit de demander sa normale à la dernière face qui a fait l'objet d'une intersection en appellant la méthode correspondante du *Gparallelog* concerné.

3.3 Intersection avec un `G6faces`

L'algorithme est relativement simple, vu que comme il a déjà été mentionné plus haut, cette figure utilise des instances de la classe `Gparallelog` pour représentation. Il suffira donc de parcourir la liste des instances de `Gparallelog` et de déterminer s'il y a intersection et de déterminer, parmi les intersections trouvées, celle se trouvant le plus près du point de départ du rayon.

3.4 Représentation d'un `Gcylinder`

Dans cette sous-section, nous allons expliquer comment nous représentons un cylindre. Nous allons d'abord rappeller brièvement la défintion d'un cylindre, puis nous expliquerons la méthode utilisée dans notre implémentation pour la construction en elle-même.

Remarque : Pour représenter un cylindre, nous nous sommes inspirer de la méthode décrite dans le livre [2] de la liste des références en fin de ce rapport. La plupart des explications, développements mathématiques ou schéma de cette sous-section proviennent de ce livre.

3.4.1 Définition d'un cylindre de révolution

En mathématique, un cylindre de révolution est une surface infinie, qui peut être définie par la donnée d'un axe et d'un rayon.
Soit D une droite dans l'espace et $r \in \mathcal{R}_+^*$ un nombre réel strictement positif. Le cylindre de révolution d'axe D et de rayon r est constitué de l'ensemble des points de $\mathcal{R}^3$ qui sont situés à une distance r de la droite D. Pour la représentation, nous définirons d'abord le cylindre comme ayant un axe de rotation coïncidant avec le troisième axe de coordonnées $\mathcal{O}_z$. Nous obtiendrons le cylindre final par l'application de transformations géométriques (translation et rotations, telles que nous les avons vues au cours).

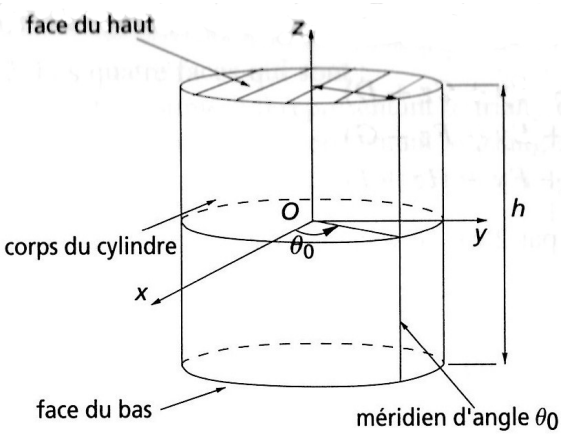
Ainsi, un cylindre de révolution dont l'axe coïncide avec l'axe Z admet une équation de la forme :

$$x^2 + y^2 = r^2 \tag{12}$$

Enfin, les cylindres que nous représentons sont limités en hauteur. Un autre paramètre à prendre en compte dans la définition du cylindre sera donc également sa hauteur $h \in \mathcal{R}_+^*$.

3.4.2 Méridiens sur un cylindre de révolution

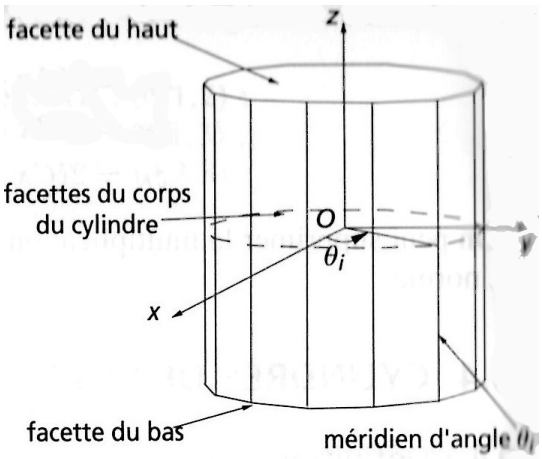
Les méridiens sur un cylindre de révolution de rayon r et de hauteur h sont les segments de droite, contenus dans le corps du cylindre, de longueur h et parallèles à l'axe Z. Comme on peut le voir sur la figure ci-dessous, un tel segment est identifié par un certain angle θ_0, avec $\theta_0 \in [0, 2\pi]$.



3.4.3 Facettisation d'un cylindre de révolution

Dans notre implémentation, nous avons choisi d'approximer un cylindre de révolution par un polyèdre.

L'idée est la suivante : étant donné un nombre de méridiens m (dans notre implémentation, nous avons fixé ce nombre à 32), nous allons considérer des méridiens $\mathcal{M}_i$ d'angle θ_i pour $i = 0, ...m$ régulièrement disposés sur le corps du cylindre. Nous construirons ensuite une facette rectangulaire entre les méridiens $\mathcal{M}_i$ et $\mathcal{M}_{i+1}$ pour $i = 0, ..., m-1$. Nous construirons enfin deux facettes pour les faces haut et bas du cylindre.



Pour construire les différentes facettes du polyèdre approximant le cylindre, nous avons besoin de connaitre tous les sommets de ce polyèdre. Nous expliquons dans le cadre ci-dessous une méthode à utiliser, et qui est celle que nous avons implémentée dans notre projet.

Nous allons construire $4m + 2$ sommets. Considérons les angles :

$$\theta_i = 2\pi \frac{i}{m} \tag{13}$$

pour i = 0, ..., m. Avec une telle variation pour i, l'angle θ_i varie régulièrement entre 0 et 2π. Soit $\mathcal{M}_i$ le méridiant d'angle θ_i.
Nous construisons tout d'abord la liste des sommets du polyèdre.

- Pour $i = 0, ..., m$, on définir le sommet P_i comme le point du méridien $\mathcal{M}_i$ qui est à la hauteur $-\frac{h}{2}$, et de coordonnées cartésiennes suivantes :

$$P_i = \left(r\cos(\theta_i), r\sin(\theta_i), -\frac{h}{2} \right) \tag{14}$$

- Pour $i = m + 1, ...2m + 1$, on définit le sommet P_i comme le point de méridien $\mathcal{M}_{i-(m+1)}$ qui est à la hauteur $+\frac{h}{2}$, et de coordonnées cartésiennes suivantes :

$$P_i = \left(r\cos(\theta_{i-(m+1)}), r\sin(\theta_{i-(m+1)}), +\frac{h}{2} \right) \tag{15}$$

Les sommets de la face du haut et de la face du bas du cylindre ne doivent pas être confondus avec les sommets du corps du cylindre. Pour cette raison, bien que ces sommets aient la même position dans l'espace, nous leur donnons des numéros différents :

- Pour $i = 2m+2, ..., 3m+1$, le sommet P_i (de la face du bas) équivaut au sommet $P_{i-(2m+2)}$.
- Pour $i = 3m+2, ..., 4m+1$, le sommet P_i (de la face du haut) équivaut au sommet $P_{i-(3m+2)}$.

Une fois que tous les sommets sont calculés, il ne reste plus qu'à les utiliser pour construire les m facettes du cylindre qui sont comprises entre deux méridiens successifs $\mathcal{M}_i$ et $\mathcal{M}_{i+1}$, puis la facette du bas et la facette du haut.
Pour représenter les facettes entre deux méridiens successifs, nous utilisons un ensemble d'objets `Gparallelog` tandis que pour les faces du haut et du bas, des `Gpolygon`.

3.4.4 Rotation du cylindre

Une fois le cylindre construit, il faut éventuellement les faire pivoter dans une certaine orientation. En effet, pour rappel, nous avons considéré jusqu'ici un cylindre de révolution dont l'axe coïncide avec l'axe Z. Dans ce but, nous avons implémenté trois nouvelles fonctions pour la rotation dans la classe `point_t` : *rotateX()* , *rotateY()* et *rotateZ()*, qui appliquent une

rotation à un point selon l'axe X, Y ou Z, respectivement, ainsi qu'une méthode de translation d'un point par un vecteur de coordonnées (x, y, z). Nous avons implémenté les trois premières transformations en utilisant simplement les matrices de rotations vues au cours théorique.

3.5 Calcul de la normale à un *Gcylinder* (en un point)

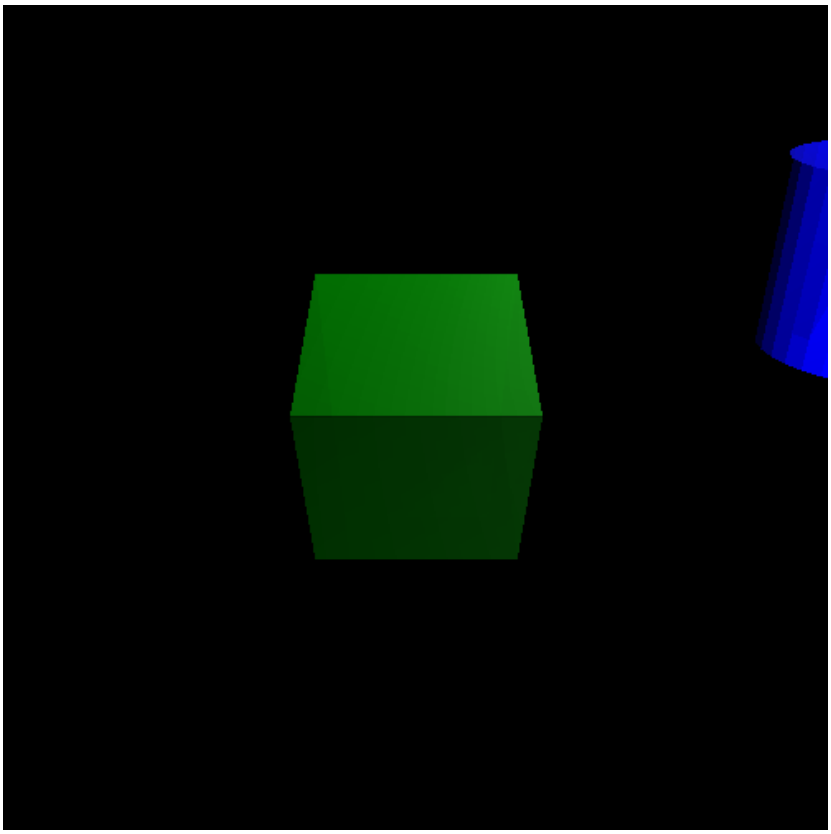
Le calcul de la normal est simplement celui de la face correspondante au point d'intersection entre le rayon et le `Gcylindre`.

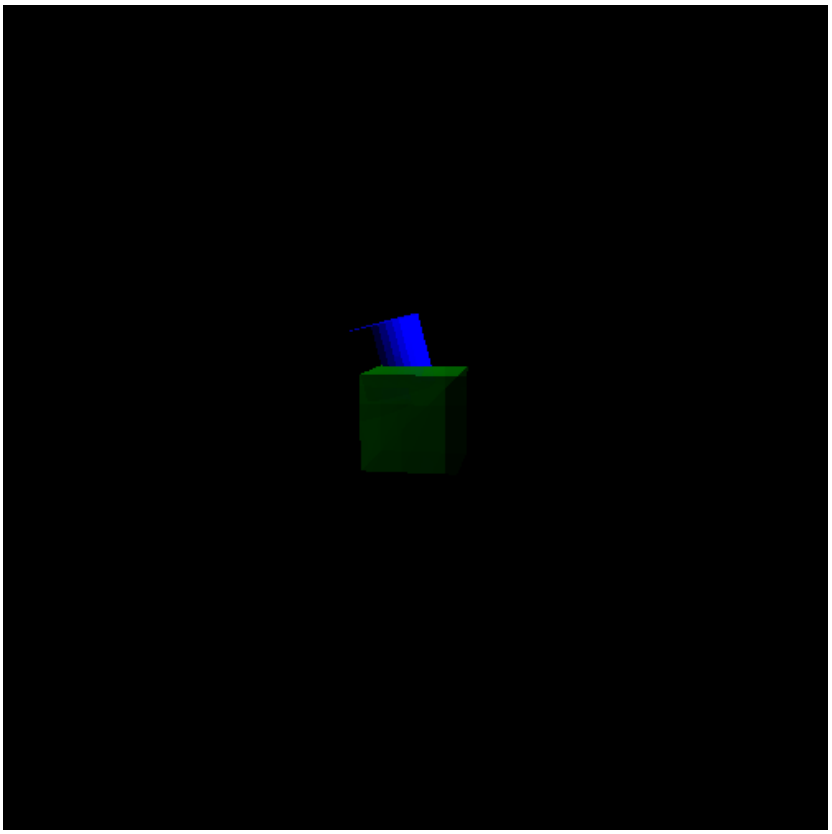
3.6 Calcul de l'intersection entre un rayon et un *Gcylinder*

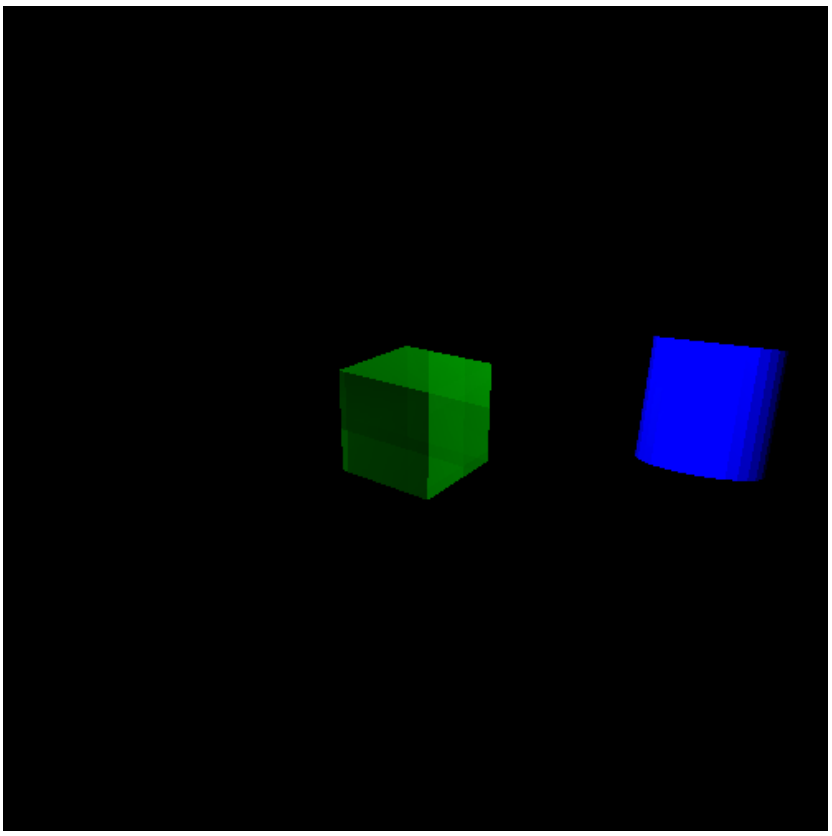
Ce calcul consiste simplement en l'appel des méthodes d'intersections des différents faces composants le `Gcylindre` (pour rappel, ces faces sont elles-même des objets `Gpolygon` ou `Gparallelog`), et parmis toutes les intersections trouvées, nous renvoyons celle qui est la plus proche du point de départ du rayon.

4 Résultats

Suivent quelques résultats de rendus obtenus avec notre version du travail :







5 Contribution

Nous avons travaillé en totale collaboration. Toutes les structures sont donc le fruit d'une discussion commune et de recherches communes.

Nous listons également les fichiers que nous avons modifié au cours de ce projet :

- *models.cc/h* : ajout des formes géométriques
- *ray.cc/h* : modification de la classe `point_t` pour permettre les transformations géométriques pour un point
- *scene.cc* et *framebuffer.cc* : modification des appels à la fonction *get_first_intersection()* des instances de *Gmodel*.

6 Conclusion

En conclusion, nous pouvons dire que ce travail nous aura fait avancer et ce sur plusieurs points. D'abord, il constitue notre première réelle expérience dans l'utilisation d'OpenGL, ensuite il nous a permis de mettre en pratique certains concepts théoriques vus au cours, et nous a donc un peu mieux préparé à une étude que si nous ne l'avions pas eu.

Références

[1] Michael T. Goodrich and Roberto Tamassia. *Data Structures and Algorithms in Java*. John Wiley & sons, 4 edition, 2006.

[2] Rémy Malgouyres. *Algorithmes pour la sythèse d'images et l'animation 3D*. Dunod, 2ème edition, 2005.

[3] Tomas Möller and Ben Trumbore. Fast, minimum storage ray/triangle intersection.