Automne 2012

LO12 PROJET FINAL

Lost in Space | Guyard Romain, Louison Céphise

Table des matières

Table des figures :

I - Introduction

Le projet final de LO12 a pour objectif d'appliquer ce qui a été vu au cours du semestre sur l'utilisation d'OpenGL dans une scène plus complexe. Nous avons décidé de développer un mini jeu vidéo en nous basant sur le moteur graphique que nous avons développé au cours des séances de TP. Notre projet propose de plonger l'utilisateur dans un univers spatial où il pourra piloter un vaisseau spatial au milieu de champ d'astéroïde.

Tout au long des séances de TP nous avons adapté les connaissances qui nous ont été apprises pour former un moteur graphique générique et facile d'utilisation. Notre projet final est donc une utilisation de notre moteur 3D.

II - Fonctionnalités de base

1) Diversité des objets de la scène

L'environnement contient plusieurs modèles 3D différents. Ils ont été créés ou modifiés avec le logiciel de modélisation Blender. Les modèles sont composés d'un ou plusieurs objets avec une ou plusieurs matières et utilise des textures.

- **Vaisseau mère :**

Le vaisseau mère est un énorme vaisseau spatial. Il a été trouvé sur internet [TS12] puis modifié pour les besoins du projet. Nous avons modifié le vaisseau pour aménager un hangar et ajouter d'une porte pour le hangar ainsi qu'un gyrophare (en dessus de l'entrée du vaisseau). La porte du hangar ainsi que le gyrophare sont animés. Lorsque l'utilisateur s'approche de l'entré la porte s'ouvre et le gyrophare démarre.

Information sur le modèle :

- Nombre de points : 3825,
- Nombre de faces : 7699.

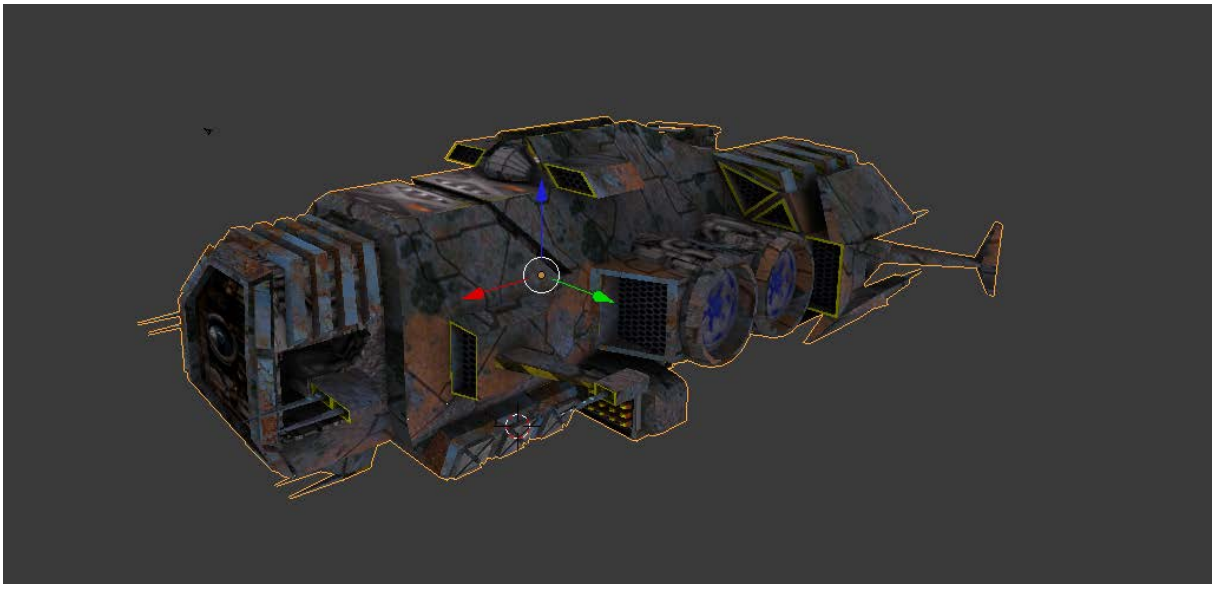

Figure 1 : Vaisseau mère

- **Vaisseau d'exploration :**

C'est un petit vaisseau que l'utilisateur peut contrôler (mode jeu). Le vaisseau a été trouvé sur internet [TS12] puis modifié. L'intérieur du cockpit a été aménagé : ajout de siège, tableau de commande et une nouvelle texture ont été fait. En mode jeu le joueur se trouve à la place du pilote. Il peut diriger le vaisseau.

Information sur le modèle :

- Nombre de points : 981,
- Nombre de faces : 1962.

Figure 2 : vaisseau d'exploration

- **Astéroïdes :**

Il y a une quarantaine d'astéroïde différent. Ils sont ajoutés dans la scène de façon aléatoire. Ils sont ensuite répartis dans l'espace aléatoirement pour former un champ d'astéroïde. Les astéroïde se déplace ensuite grace à une animation, qui les fait bouger dans l'espace et tournée sur eu même.

Information sur le modèle : (pour les 38 modèles d'astéroïdes)

- Nombre de points : 22956,
- Nombre de faces : 45760.

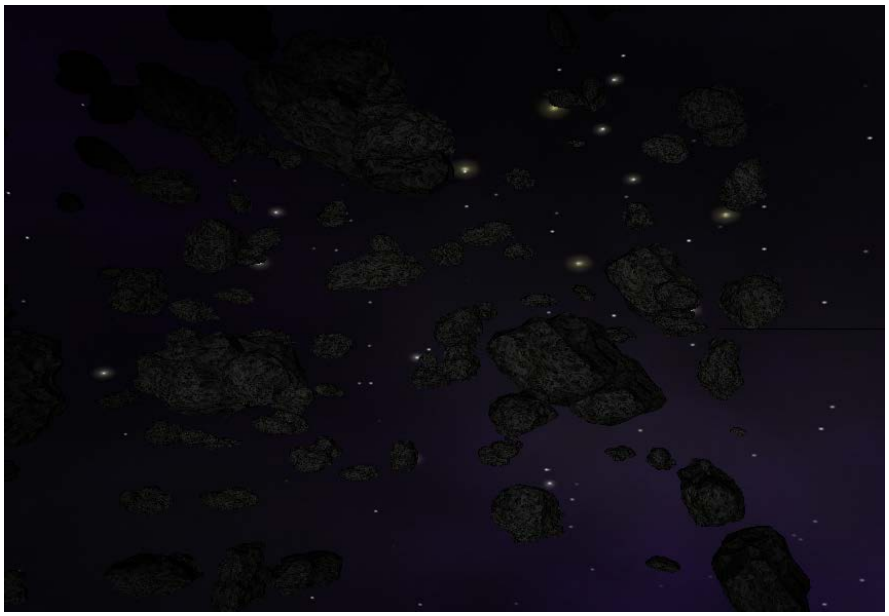

Figure 3 : champ d'astéroïdes

- **Robot :**

Un robot (appelé Robien) a été entierrement modélisé et texturé. Le joueur doit essayer de le récuperer au milieu du champs d'astéroide et le ramener dans le vaisseau mère. Robien à une animation pesanteur qui le fait dériver dans l'espace et tournée sur lui-même.

Information sur le modèle :

- Nombre de points : 3918,
- Nombre de faces : 7622.

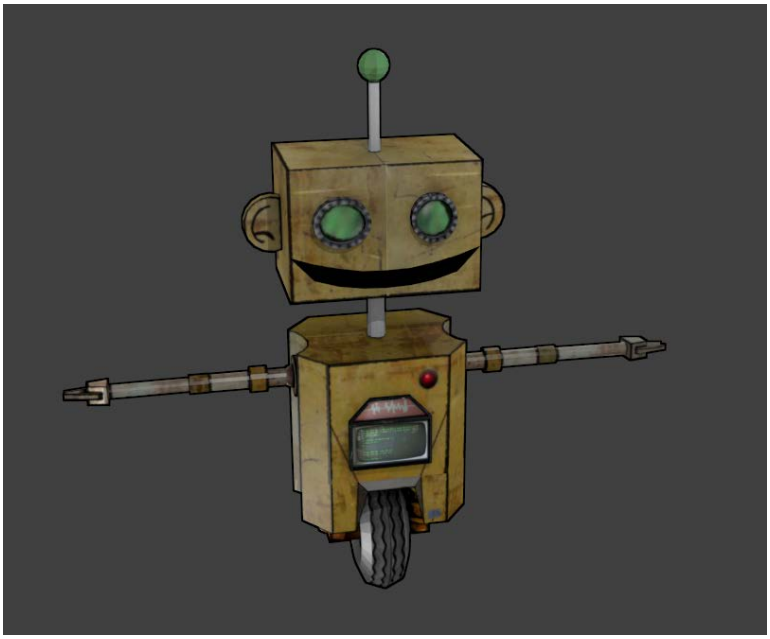

Figure 4 : Robien

- **Skybox :**

La skybox utilisée permet d'immerger l'utilisateur dans l'espace avec un fond spatial. Elle a été réalisée avec le logiciel Spacescape [SS12].

2) Objets mobiles

- **Objet déplaçable**

La plupart des objets de la scène peuvent être déplacés par l'intermédiaire des touches définies en TD : F1, F2 et F3 pour les translations suivant les trois axes et F4, F5 et F6 pour les rotations (avec la touche Shift pour les mouvements inverses).

- **Objets mobiles particuliers :**

- Porte du vaisseau mère : la porte du vaisseau mère s'ouvre et se ferme lorsque le vaisseau d'exploration s'approche du vaisseau mère. Le mouvement (rotation) se fait suivant un axe que nous pouvons définir pendant la modélisation sous Blender, et passé au moteur 3D par le fichier exporté de Blender. De plus au moment de l'ouverture de la porte un gyrophare composé de deux lumières tournant sur elle-même (rotation selon z) se déclenche au-dessus de la porte.
- Astéroïdes et Robien : les astéroïdes et le robot Robien se déplacent et tourne sur eu même de façon aléatoire (rotation et translation selon des axes aléatoire).
- Vaisseau d'exploration : l'utilisateur peut déplacer le vaisseau lorsqu'il est dans le cockpit (mode jeu) à l'aide de la sourie : clic gauche pour avancer, clic droit pour ralentir, clic molette pour stabilisation du vaisseau. Lorsque le joueur accélère l'interface utilisateur montre ca vitesse. S'il atteint une vitesse particulière, des particules de fumé apparaissent dans le vaisseau avec un message de surchauffe du vaisseau.
- Robien : Robien à une lampe qui le suit elle est à l'intérieur et éclaire comme si son écran (qui est sur son « ventre ») gênerait de la lumière.

3) Sources lumineuses

Les sources peuvent être activées et désactiver à l'aide des touches F9 (spot 3) F10 (spot 1) et F11 (spot 2)

- Spot 1 : éclairage du vaisseau, il se déplace avec le vaisseau d'exploration.
- Spot 2 : le spot peut être déplacé comme un objet normal.
- Spot 3 : source de lumière à l'infini

4) Points de vue

Plusieurs modes de caméra peuvent être utilisés, il est possible de changer de l'un à l'autre grâce à la touche 'Tabulation'.

- Caméra vaisseau : (mode jeu) vue de l'intérieur du cockpit du vaisseau d'exploration, la vue se déplace avec le vaisseau. Le déplacement se fait avec la souris.
- Caméra LookAt : les touches définies en TP,
- Caméra PilotView : les touches définies en TP,
- Caméra PolarView : les touches définies en TP,

III - Améliorations additionnelles

Nous avons ajouté beaucoup d'autre fonctionnalité pour essayer de nous rapprocher des possibilités des moteurs 3D actuel.

1) Moteur 3D en C++

Le moteur 3D est entièrement codé en programmation orientée objet (C++) notamment pour bénéficier du polymorphisme. Nous sommes parties du code donné au début des séances de TP et nous avons peu à peu ajouté les fonctionnalités (Scène, Lumière, Interaction, Caméra…) directement sous forme d'objet. Cette modification nous a permis d'avoir un projet générique. Nous avons largement utilisé les possibilités du polymorphisme, notamment pour notre graphe de scène ou nous pouvons ajouter à la fois des lumière, des Objets 3D ou des système de particule…

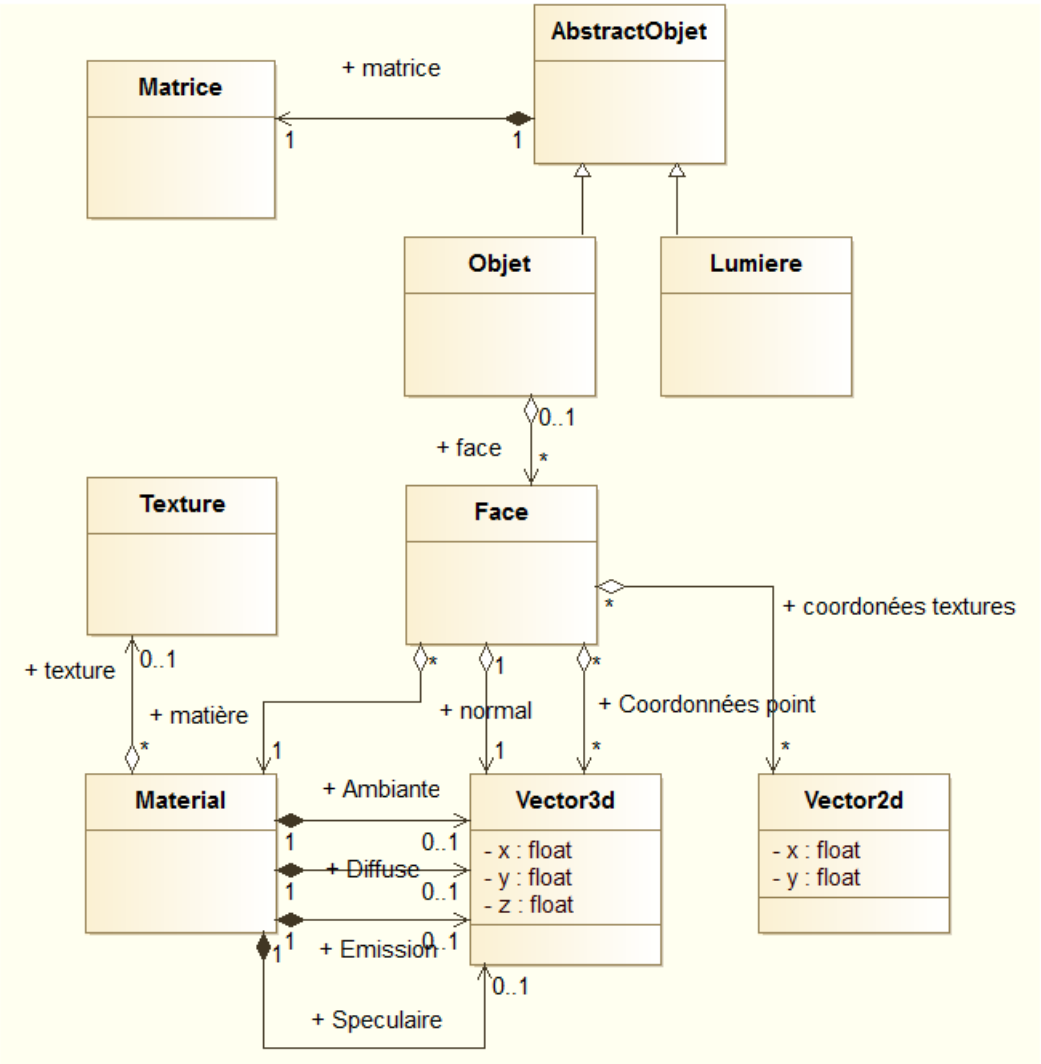

Figure 5 : Diagramme de classe simplifié des Objets

Par exemple pour représenter les Objets 3D nous avons créé une Classe *Objet*, elle contient une liste d'objets *Face*, elles-mêmes contiennent une liste de point (*Vector3d*) pour les points de l'objet, de point de coordonnée de texture (*Vector2d*) et un objet *Material* (Figure 5). Les Objets ont aussi d'autre attribut comme des animations…

2) Graphe de Scène

Le graphe de scène structure de manière logique la représentation d'une scène graphique. Le graphe de scène est une collection de nœuds *AbstractObjet* (Figure 5). Un nœud peut avoir plusieurs nœuds fils (*AbstractObjet*). Ainsi, un effet appliqué sur la matrice d'un nœud se répercute sur ses

descendances, ce qui permet de propager l'opération aux nœuds fils. Par exemple le vaisseau mère à comme nœud fils ça porte et le gyrophare. Si l'on effectue une rotation sur le vaisseau, cette rotation sera aussi appliquée sur la porte et au gyrophare, mais on peut aussi appliquer indépendamment une rotation sur la porte (pour l'ouvrir par exemple). Toutes les classes héritées d'*AbstractObjet* peuvent être mis dans le graphe de scène. C'est le cas des Objet 3D, des systèmes de particule, des lumières…

3) Brouillard

Un effet brouillard (Objet *Brouillard*) peut être ajouté. Les objets loin de l'observateur seront peu visibles et plus l'utilisateur se rapprochera plus il les verra. Cet effet est créé grâce à l'option GL_FOG d'openGL.

4) Gestion des collisions

Pour que le jeu sois plus intéressant ou avons ajouté la gestion des collisions. La gestion des collisions se fait en deux temps. Dans un premier temps la collision est effectuée entre les sphères englobant les objets de la scène. Si une collision est détectée, un deuxième niveau de détection de collision plus précis est calculé entre un point et la sphère (pour chaque face nous calculons son centre de gravité). Actuellement cette option présente encore quelques problème nous avons décidé de ne pas activer cette option.

5) Gestion des déclencheur/ trigger

Pour détecter des évènements et pouvoir lancer des animations en conséquence, nous avons mis en place un système de trigger qui détecte quand un élément est proche d'un autre par exemple. Cette fonctionnalité utilise donc des composants développés pour le gestionnaire de collision. Nous l'avons par exemple utilisé pour la porte qui s'ouvre lorsque le vaisseau de l'utilisateur s'approche de l'entrée puis se referme une fois celui-ci loin.

6) Vertex Buffer Object (VBOs)

L'utilisation du « mode immédiat » d'opengl (utilisation de glBegin, glEnd) est peu performante et désormais obsolète. L'utilisation des VBOs permet d'obtenir de bien meilleures performances. Les données 3D (vertex, normales, face, couleurs, point de texture) sont envoyées au début du programme vers carte graphique. Ces données ne résident plus dans la mémoire système, mais dans la mémoire de la carte graphique ce qui permet un rendu plus rapide.

Cette optimisation améliore considérablement les performances et nous permet de mettre un plus grand nombre d'objet et plus détaillé.

7) Import de fichier OBJ et MTL

Nous avons décidé de créer notre propre importeur de fichier de modèle 3D. Nous avons choisi d'utiliser les formats de fichier OBJ pour décrire les géométries 3D et MTL pour les matières. Ces formats de fichier sont au format ASCII et ils sont ouverts [OBJ12]. Ils sont utilisés par la plupart des logiciels de modélisation 3D. Notre importeur permet de lire les informations des fichiers OBJ et de directement implémenter des objets de la classe *Objet*. Il lie aussi les objets à leurs matières en ouvrant les fichiers MTL et en créant des objets Matériels. L'importeur utilise le Png loader vu en TP réadapté sous forme objet pour ouvrir les images utilisées en textures. Nous avons utilisé la modularité de l'OBJ pour permettre d'importer d'autre information comme des axe de rotation

utilisé ensuite pour les animations (comme pour la porte). Nous pourrions à terme imaginer d'animer un squelette (bras du robot Robien par exemple) avec cette technique.

8) Interface utilisateur et texte

Nous avons à partir des options 2D d'OpenGL et de Glut créé des *affichages tête haute* (*Head-up display – HUD*). Ils nous permettent d'afficher des informations à l'écran pour l'utilisateur tel que du texte ou des textures. Pour le projet, nous l'utilisons notamment pour mettre un viseur au centre de l'écran, un indicateur de vitesse ou encore les instructions du jeu. Pour pouvoir les voir en toutes circonstances, il faut penser à désactiver l'éclairage avant de les afficher. Nous pouvons mettre des mettre des texture et/ou de la couleur sur les éléments de l'interface et nous pouvons les animés comme c'est le cas du compteur de vitesse.

9) Utilisation de shader et modèle de Phong

Les modèles d'ombrage Flat et de Gouraud ne donnant pas des rendus très réalistes, nous avons décidé d'utiliser le modèle d'illumination de Phong. Pour mettre en place ce modèle d'illumination nous utilisons un shader.

Le shader à été programmé en GLSL. Il se sépare en deux programmes distincts. Le premier, appelé « Vertex Shader » (shader de sommet) est exécuté pour chaque sommet de chaque objet. Ici, il sert à calculer les vecteurs d'incidences. Le deuxième, le « pixel shader » (ou « fragment shader ») est lui exécuté pour chaque pixel de l'image final. Le shader éclaircit ou assombrit la couleur de la texture suivant les règles d'éclairage du modèle de Phong. Il est possible de passer d'un modèle d'éclairage à l'autre avec la touche 'S'.

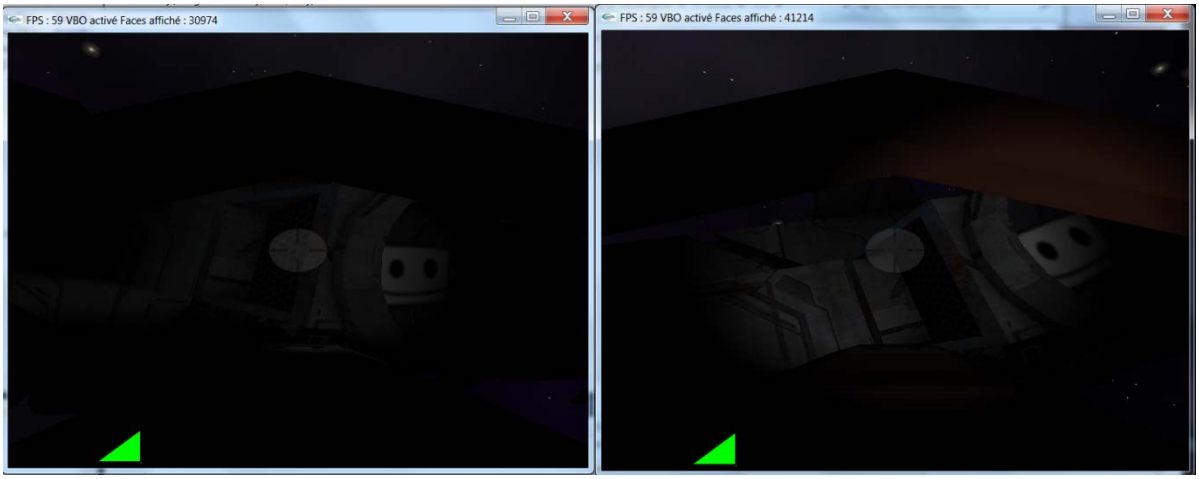

Figure 6 : Modèle de Gouraud et modèle de Phong

10) Système de particule

Nous avons ajouté un système de particule. Une particule est formée de deux triangles et à une durée de vie, une couleur. Le système de particule crée un très grand nombre de particules (supérieur à 1000 pour un rendu correct). Il permet de faire des jets d'eau, des explosions, de la fumée… Nous avons utilisé pour faire de la fumée quand le vaisseau d'exploration est en surchauffe.

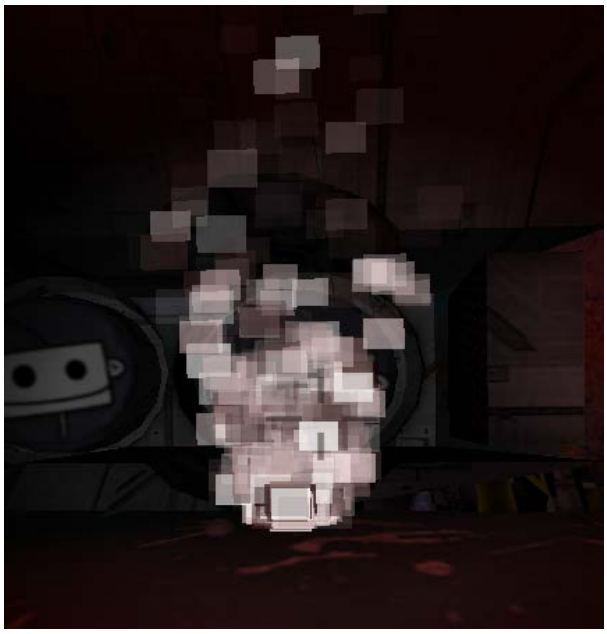

Figure 7 : Système de particules

11) Occlusion culling

L'occultation des parties cacher permet d'améliorer les performances en « supprimant » les objets non visibles des calculs de la carte graphique. Pour détecter si un objet est visible, nous testons s'il est face à l'utilisateur ou non.

12) Mode fileté

Il est possible d'activer le mode fileté, il permet de voir seulement les arêtes des objets. Il faut utiliser la touche 'F' pour activer et désactiver cette option.

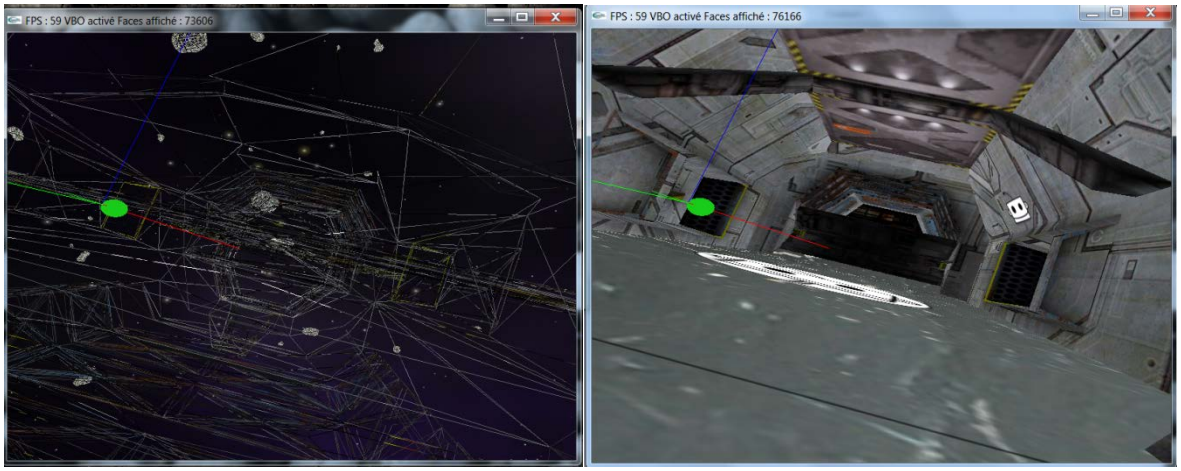

Figure 8 : différence entre mode filté et normal

IV - Référence :

[TS12] Bibliothèque de modèle 3D, URL : http://www.turbosquid.com/ , 2012

[SS12] Création de skybox spatial, URL : http://alexcpeterson.com/portfolio/spacescape, 2012

[OBJ12] Spécification des fichiers OBJ, URL : http://people.sc.fsu.edu/~jburkardt/txt/obj_format.txt, 2012