Projet Image : Conversion d'objets 3D triangulés en objets discrets

Baud Thierry - Guizani Djemel
ESIL - Informatique 3ième Année

26 février 2002

Table des matières

1 Introduction

Les *volumes numériques* sont l'analogue en 3D d'une image informatique 2D classique : un tableau à trois dimensions (au lieu de deux) contenant en chaque point, appelé voxel (au lieu de pixel), une valeur qui code le niveau de gris. Quand le volume ne contient que deux niveaux de gris pour ses voxels (1 ou 0), on parle alors de *volume binaire*.

Des travaux menés dans l'équipe LXAO ont pour but d'analyser puis de reconstruire une forme 3D contenue dans un tel volume binaire. Ces travaux se heurtent à une difficulté simple mais ennuyeuse : de telles images sont quasiment toujours soumises au secret médical ou à des copyrights. Il est donc très difficile de se constituer un jeu d'exemples de volumes numériques pour effectuer des démonstrations, alors qu'il en existe en revanche de nombreux pour les objets 3D triangulés et pour les images 2D.

Notre travail a consisté à développer une application qui à partir d'un objet 3D triangulé produit une *volume binaire.* Deux formats d'objet 3D sont supportés : les Objets LightWave (.LWO) et les Objets 3DStrudio (.3DS). Il est ensuite possible de sauvegarder les *volumes numériques* sous forme fillaire, en face pleine et en volume plein.

Cette application a été développée en C++ sous Linux avec les librairies LibDv et OpenGL. Les *volumes numériques* générés peuvent être visualisés et manipulés avec l'outil GtkDv.

2 Objets 3D Triangulés

L'intérêt principal de l'utilisation du langage C++ est la possibilité d'évolution de l'application. Nous avons donc mis en place une structure de classe pour les objets 3D triangulés qui permettra le support de différents formats de fichiers. Actuellement les fichiers .LWO et .3DS sont supportés.

2.1 Structure Principale

Tous les objets 3D triangulés, quelque soit leur format, héritent d'une classe simple *ObjetFace*. Ces objets sont constitués d'une liste des faces. Chaque face est une liste de sommets, une couleur codée en RGB de 0 à 255 et un vecteur normal. Même si nous travaillons avec des objets appelés 3D triangulé, le nombre de sommets des différentes faces n'est pas limité à 3. Lorsque cette structure est remplie, il est assez simple de l'afficher avec OpenGL grâce *GL_ TRIANGLES*, *GL_ QUADS* ou *GL_ POLYGON*. C'est dans cette classe que nous calculons la boîte englobante de l'objet qui nous

permet de faire la mise à l'échelle. C'est à partir de cette structure que nous générons les volumes numériques.

2.2 Les Objets LightWave

Les objets LightWave ont une structure assez simple. Les fichiers .LWO doivent commencer par les deux marqueurs *FORM* et *LWOB*. On trouve à la suite de *FORM* le nombre de bytes à lire.

Ensuite, on trouve les marqueurs *PNTS*, *SRFS*, *POLS* et *SURF* suivis d'information. La seule condition sur l'ordre des marqueurs est que *PNTS* et *SRFS* doivent être situés avant *POLS* et *SURF*.

On trouve à la suite du marqueur :

- *PNTS*, la liste des coordonnées X, Y et Z de tous les points de l'objet. On stocke ces informations dans une liste de coordonées
- SRFS, la liste des noms des surfaces de l'objet, par surface on entend les différents matériaux, donc dans notre cas des couleurs. On stocke ces informations dans une liste de couleur arbitrairement fixées à $(R, G, B) = (180, 180, 180)$.
- *POLS*, la liste des polygones des l'objets. C'est en fait le nombre de sommets puis une liste d'indices correpondants aux points lus après le marqueur *PNTS*. La dernière information est l'indice de la surface qui permettra de fixer la couleur du polygone. On remplit la structure principale : la liste de face.
- *SURF*, les informations d'une surface prédéfinie après le marqueur *SRFS*. On se contente pour ces marqueurs de lire les paramètres R,G,B. On complète alors la liste de couleur et la structure principale.

Objet Lightwave visualisé avec Obj2Dv

2.3 Les Objets 3DStudio

Un fichier au format 3D Studio est formé de blocs de données appelés "Chunks". Chaque chunk est formé de :

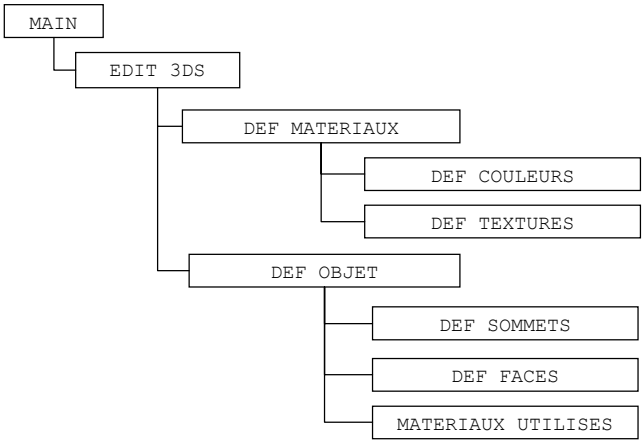
- Un identifiant : ID
- Sa taille
- les données à interpréter.

Si on ne connait pas la signification du chunk ou si l'on ne veut pas l'interpréter il suffit de déplacer le pointeur de lecture.

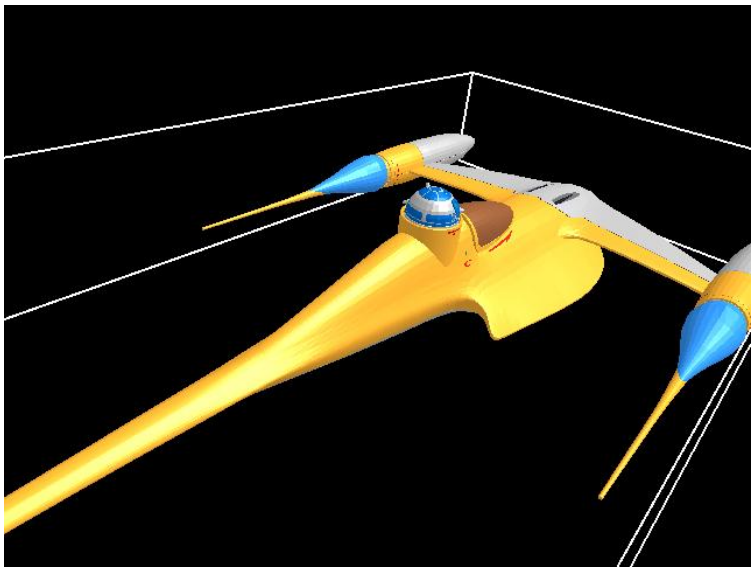
Pour notre application nous devions charger l'ensemble des objets contenus dans la scène 3DS avec leurs couleurs sans se préoccuper des animations et de toutes les définitions des matériaux texturés.

La grande difficulté dans la lecture de ces fichiers résidait dans l'imbrication des blocs. Il a donc fallut mettre en place un ensemble de fonctions de lectures dépendant de la position du pointeur dans le fichier. Ici nous avons donc créer les fonctions de lecture des Materiaux, des Objets et de l'arborescence principale.

Comme il peut y avoir plusieurs objets dans un fichier 3D Studio, nous avons décidé de les concaténer en un seul et même objet dans notre structure principale.



Arborescence simplifiée d'un fichier au format 3D Studio



Objet 3D Studio visualisé avec Obj2Dv

3 Volumes Binaires

Nous avions, grâce à la librairie LibDv, accès à un ensemble de fonctions et de structures qui nous ont permis de manipuler des volumes numériques. Nous avons intégrer ces éléments dans une classe C++, *ObjetDv*.

3.1 Structure Principale

Un *volume numérique* est créé à partir d'un ObjetFace et d'une taille fixée par l'utilisateur. On peut sauvegarder le volume numérique dans un fichier à

n'importe quelle étape de sa création (fillaire, faces pleines, volume plein).

3.2 La Méthdode de Bresenham 3D

Pour pouvoir discrétiser un segment de droite dans un volume numérique, il nous a fallut mettre en place un algorithme de Bresenham généralisé à 3 dimensions.

Après quelques recherches sur Internet, nous avons développé un algorithme proche de la méthode de Bresenham en 2 dimensions.

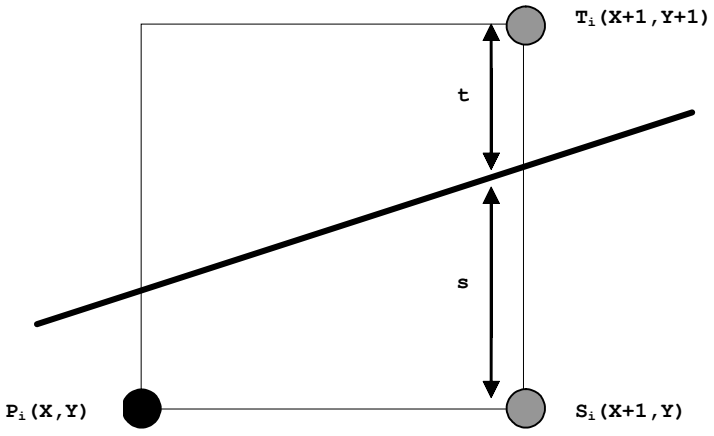
Algorithme Ligne3D :

```
Tester la coordonnee sur laquelle on progresse le plus

Exemple pour X :
Faire de Xmin a Xmax
  Avancer de 1 sur X
  Tester si progression sur Y
    Si progression alors Avancer de 1 dans la direction de Ymax
  Tester si progression sur Z
    Si progression alors Avancer de 1 dans la direction de Zmax

Boucle identique pour Y ou Z dominant
```

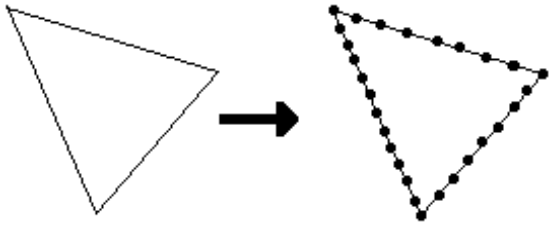
Nous avons bien sur un algorithme optimisé qui n'effectue pas de traitement sur des flottants mais uniquement sur des entiers. De plus, les divisions et les multiplication par 2 se font par décallage.



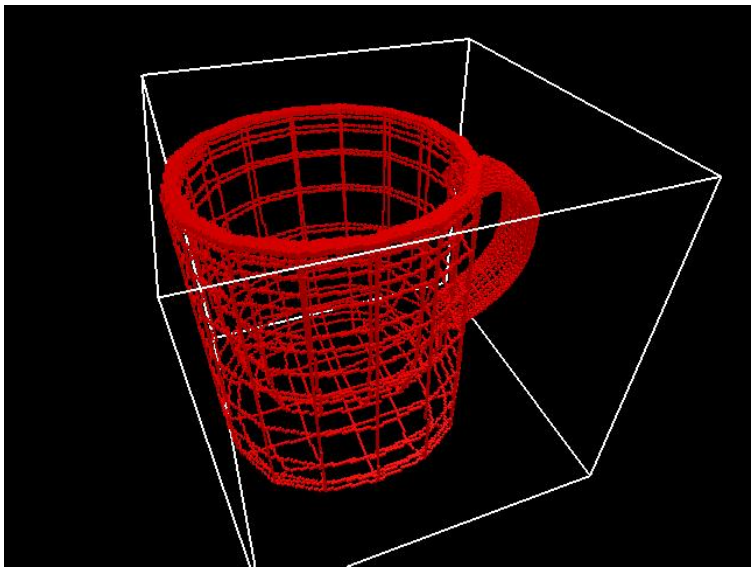
Critère de choix : $s - t < 0$ choisir S_i sinon T_i

4 Représentation en Fillaire

Par défaut, on crée "l'image" sous forme fillaire de l'objet 3D triangulé dans le volume binaire. Pour chacun des segments des faces de l'objet, on fait une mise à l'échelle dans la matrice représentative du volume et on dessine une ligne 3D par la méthode de Bresenham.



La première étape de génération des volumes munériques



Volume binaire en filaire générer à partir du modèle 3D

Algorithme de création du volume numérique en fils de fer :

```
Pour chaque face du modele 3D triangule
{
  Pour chaque sommet de la face
  {
    Prendre les coordonnees du sommet
```

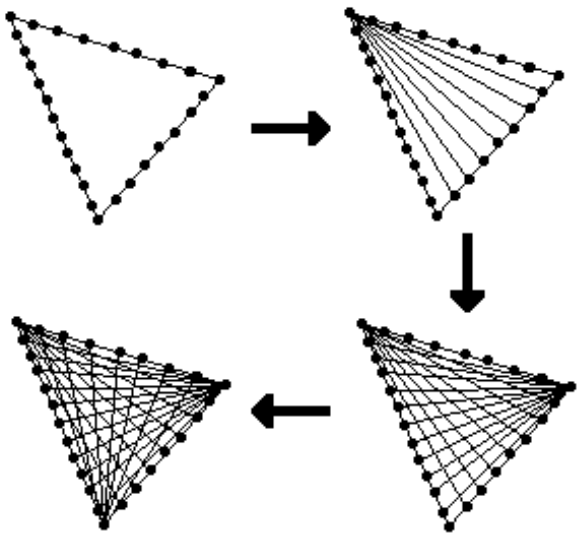
```
    Si on est au dernier sommet
     Prendre les coordonnees du premier sommet
    Sinon
     Prendre les coordonnees du sommet suivant

    Mise a l'echelle des coordonnees
    Allumer les points du segment qui relie les 2 sommets
  }
}
```

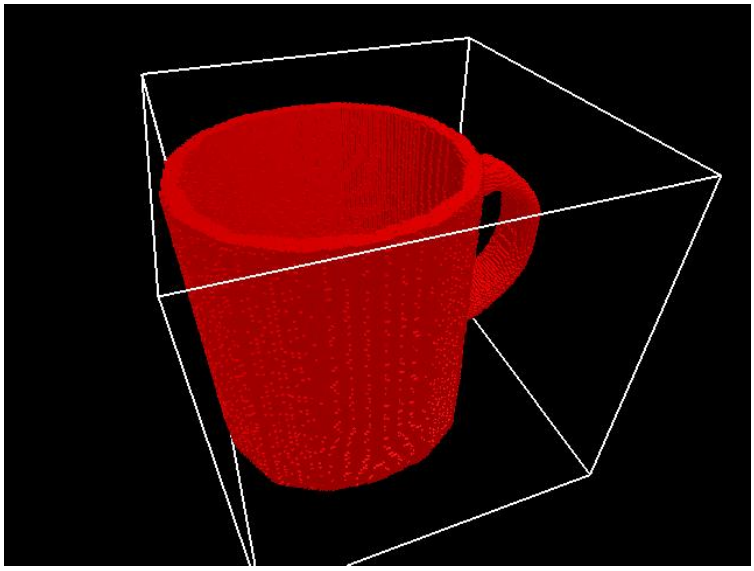
4.1 Remplissage des Faces

Nous n'avons malheureusement pas réussi à trouver une méthode optimale de remplissage des faces. A chacun de nos essais par différentes méthodes nous avons obtenu, suivant la forme des faces, des trous. Nous avons alors choisi une méthode lourde mais efficace (mieux vaut trop remplir que pas assez !).

Pour chacune des faces, nous divisons dans un premier temps en triangles, puis nous remplissons ceux-ci : La méthde de Bresenham 3D appliquée à chacun des 3 segments des triangles obtenus nous donne 3 listes de points. Nous traçons ensuite les lignes 3D entre un sommet du triangle et les points de du segment opposé.



La méthode de remplissage des faces



Volume binaire en faces pleines générer à partir du modèle 3D

Algorithme de remplissage des faces :

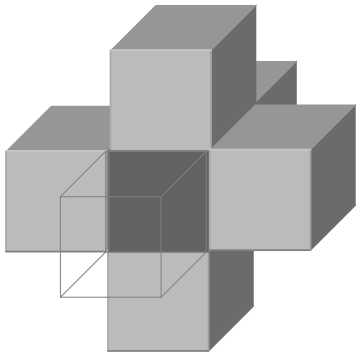
```
Triangulation de chaque face du modele 3D
Pour chaque triangle
{
  Pour chaque point du premier segement
  {
    Allumer les points entre le sommet oppose
      et le point du premier segement
  }
  Pour chaque point du second segement
  {
    Allumer les points entre le sommet oppose
      et le point du second segement
  }
  Pour chaque point du troisieme segement
  {
    Allumer les points entre le sommet oppose
      et le point du troisieme segement
  }
}
```

4.2 Remplissage du Volume

Nous avons mis en place deux méthode de remplissage de volume.

4.2.1 Méthode Récursive par 6-Connexité

Pour cette méthode, l'utilisateur doit fournir à l'application les coordonnées d'un ou plusieurs points de départ. Ensuite, l'application se charge de remplir le volume en s'arrêtant juste sous la surface de l'objet.

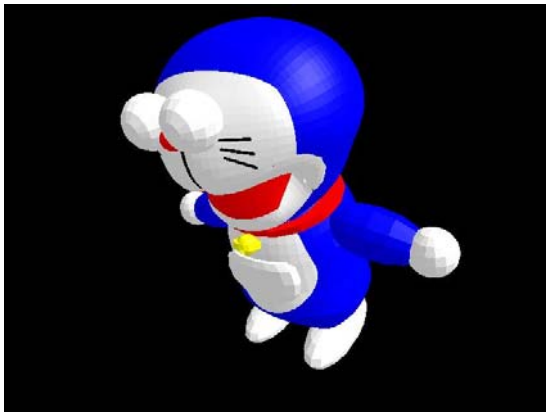


La 6-connexité d'un voxel

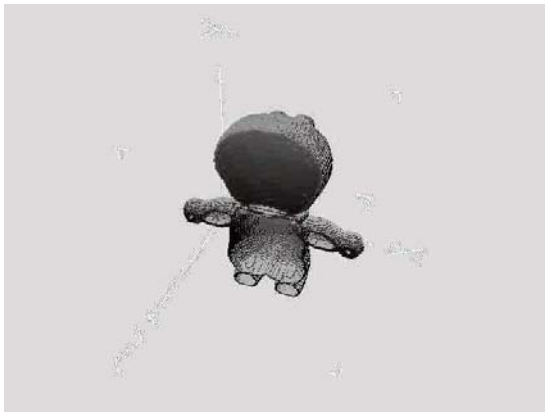
Algorithme de remplissage récursif par 6-connexité :

```
Generer le modele avec les faces pleines
Pour chaque point eteind de coodonnees X,Y,Z donne
  Allumer 6-connexite(X,Y,Z)
....

Allumer-6-Connexite(X,Y,Z)
  Allumer le point en X,Y,Z
  Si le point en X+1,Y,Z est eteind
   Allumer-6-Connexite(X+1,Y,Z)
  Si le point en X-1,Y,Z est eteind
   Allumer-6-Connexite(X-1,Y,Z)
  Si le point en X,Y+1,Z est eteind
   Allumer-6-Connexite(X,Y+1,Z)
  Si le point en X,Y-1,Z est eteind
   Allumer-6-Connexite(X,Y-1,Z)
  Si le point en X,Y,Z+1 est eteind
   Allumer-6-Connexite(X,Y,Z+1)
  Si le point en X,Y,Z-1 est eteind
   Allumer-6-Connexite(X,Y,Z-1)
```



Le modèle 3D triangulé original



Vue en coupe sous GtkDv du volume plein : remplissage à partir d'un point situé dans la "tête" du modèle

4.2.2 Méthode Automatique

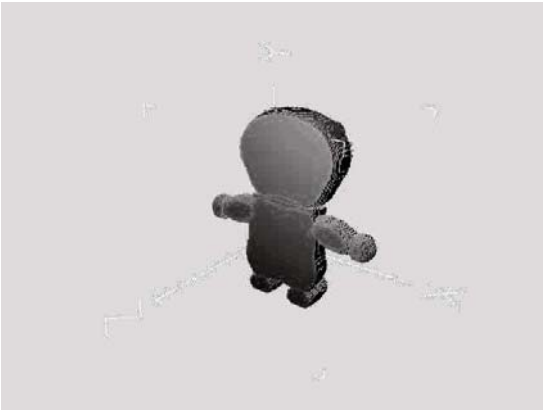
L'autre méthode que nous avons mise en place est basée sur une fonction membre de la classe ObjetDV. Cette fonction *Interieur(x,y,z)*, renvoie vrai ou faux suivant si le voxel passé en paramètre est à l'intérieur de l'objet ou non. Pour le remplissage, on parcourt tout le volume numérique et on allume les points *intérieurs*. La méthode *Interieur(x,y,z)* lance à partir du voxel passé en paramètre, 6 rayons dans les directions de faces (6-connexité), si tous les rayons touchent la surface de l'objet, on considère que le voxel est à l'intérieur de l'objet.

Mais, cette définition de l'intérieur de l'objet n'est pas complète, il arrive que des voxels qui sont extérieurs à l'objet soient allumés.

Cette méthode s'avère efficace pour des objets ou des assemblage d'objet simples. Mais le remplissage récurssif par 6-connexité est plus sûr.

Algorithme de test pour l'intérieur :

```
booleen Interieur(X,Y,Z)
  Lancer un rayon de X a DV->Max.X
  Si le rayon intersecte avec la surface alors continuer
  Sinon renvoyer faux
  Lancer un rayon de X a 0
  Si le rayon intersecte avec la surface alors continuer
  Sinon renvoyer faux
  Lancer un rayon de Y a DV->Max.Y
  Si le rayon intersecte avec la surface alors continuer
  Sinon renvoyer faux
  Lancer un rayon de Y a 0
  Si le rayon intersecte avec la surface alors continuer
  Sinon renvoyer faux
  Lancer un rayon de Z a DV->Max.Z
  Si le rayon intersecte avec la surface alors continuer
  Sinon renvoyer faux
  Lancer un rayon de Z a 0
  Si le rayon intersecte avec la surface alors continuer
  Sinon renvoyer faux
  Renvoyer vrai
```



Vue en coupe avec GtkDv du volume plein : remplissage de tous les volumes internes

5 L'Application "Obj2Dv"

5.1 Manuel Utilisateur

5.1.1 Démarrage

L'application Obj2Dv se lance en ligne de commande depuis n'importe quel type de terminal Linux (UNIX). Il faut absolument donner un argument qui soit un fichier objet. Pour le moment notre application support les fichiers objets au format LightWave (.LWO) et au format 3D Studio (.3DS).

Lorsque l'application démarre, une fenêtre OpenGL contenant la vue de l'objet 3D triangulé. On peut modifier la vue en déplaçant la souris avec le boutton gauche pressé.

5.1.2 La Vue 3D Triangulé

Cette vue nous montre l'objet 3D, après la lecture du fichier et le stockage dans la structure de liste de faces définie plus haut. On peut grâce au menu, créer des fenêtres de vue de volume numérique de différente taille (50 - 300) et quitter le programme.

5.1.3 La Vue de Volume Binaire

Un fois que l'on a sélectionner la taille de volume numérique de l'on souhaite, une nouvelle fenêtre OpenGL apparaît avec l'objet représenté en fillaire. Le menu nous permet alors de remplir les faces, le volume suivant les deux méthodes décrites précedement, de sauver le volume binaire et de quitter le programme. Le menu Vue permet d'afficher les voxels allumés du volume numérique sous forme de points simples ou de petits cubes.

A tout moment, on peut obtenir la représentation que l'on souhaite, les voxels sont recalculés à chaque fois.

5.1.4 Les touches claviers disponibles

Dans toutes les fenêtres OpenGL :

- Touches 'a' et 'z' : Affichage des axes de l'objet
- Touches 'w' et 'x' : Zoomer en avant ou en arrière vers l'objet
- Touches 'b' et 'n' : Affichage de la boîte englobante de l'objet
- Touches 'e' et 'r' : Activer/Désactiver la gestion OpenGL des faces cachées
- Touches 'd' et 'c' : Activer/Désactiver la gestion OpenGL du Z-Buffer

– Touches 'l' et 'm' : Activer/Désactiver la gestion OpenGL de la lumière
– Touches 'j' et 'k' : Régler l'intensité de la lumière.

5.2 Les "petits" Problèmes

Nous avons développé notre application sur plateforme Linux avec XFree 4 (GLX) et des librairies OpenGL optimisées pour les cartes graphiques de nos configurations (NVidia). Nous avons aussi utilisée les librairies C/C++ strandard pour la manipulation des chaînes de caractères et des *"streams"* (écran ou fichier).

Sur ces configurations, tout se déroule bien, mais lorsque nous avons testé notre application sur plateforme SGI :

– Certaines normales de l'objet 3D Triangulé sont à l'envers, ainsi la lumière n'est pas correctement renvoyée et on observe des faces toutes blanches.
– L'utilsation de *ifstream.read(_)* et *ifstream.get(_)* est différente, ainsi après modification la lecture des fichiers .LWO fonctionne encore, mais les fichiers .3DS ne fonctionnent plus.

Ne disposant pas de plateforme SGI, ni du temps nécessaire pour corriger ces bugs, l'application a été laissée dans sa version utilisable sur plateforme Linux.

Un dernier problème, lors du remplissage récursif par 6-connexité d'un volume numérique d'une taille de 200 et plus, la machine plante : la récursivité explose.

5.3 Le Futur

La première des améliorations pour cette application est le portage complet sur différentes plateforme (SGI, Windows, Mac...)

Le support d'autre format de fichier est possible, il suffit seulement que le et les objets contenus dans le fichier soit de type 3D triangulé.

Il faudrait déterminer une méthode optimale de remplissage de face.

La méthode de remplissage automatique peut être améliorée en remettant à 0 les voxels allumés inutillement grâce à une 6-connexité à l'extérieur de l'objet (avec 0,0,0 comme point de départ).

6 Conclusion

Ce projet nous a permis de traiter différents points d'infographie étudiés en 3ième année :

- Analyser des formats de fichier de modèles 3D
- Mettre en place l'algorithme de discrétisation de Bresenham en 3D
- Réfléchir sur des algorithmes appliqués à des volumes numériques (remplissage facette, volume)
- Structurer l'application grâce au langage C++

Nous espérons que notre application sera utilisable dans le cadre des recherches menées dans l'équipe LXAO et notamment par Eric Rémi qui nous a encadré tout au long de ce projet.