Rapport du TER FMIN200
du Master IFPRU Informatique
du 19 Janvier au 18 Mai 2009

GROUPE A

Navigation interactive dans un paysage urbain

Powered by LaTeX

Encadrant : Nancy RODRIGUEZ

Etudiants : BENABADJI Mehdi
DELMAS Gabriel
LOMBARD Matthieu
OCHMAN Cécile

Remerciements

Nous souhaitions remercier vivement notre encadrante, Mme. Nancy RODRIGUEZ. Elle a su se montrer très disponible chaque semaine et conciliante en montrant de la patience et de la compréhension devant nos différents problèmes de réalisation de notre projet. Elle a su nous rassurer devant nos difficultés à gérer des outils encore très peu connus par nous tous en nous proposant de nouvelles pistes très intéressantes. Elle nous a également encouragé très tôt à écrire le rapport, et mettre en place un journal de bord facilitant grandement sa rédaction.

Nous tenons également à remercier Mme Mountaz HASCOËT pour venir nous soutenir lors de notre soutenance afin de palier l'absence de notre encadrante et de nous avoir encourager également.

Nous désirons remercier Benoit LANGE qui a fait preuve de beaucoup d'intérêt quant à l'avancée de notre travail.

Merci enfin à tous nos enseignants de l'université des sciences Montpellier II pour leur précieuse assistance pédagogique.

Table des matières

1 Sujet

1.1 Contexte

Tout d'abord, définissons ce qu'est un LOD. Les niveaux de détail (Level Of Details) permettent de simplifier un objet en fonction de la distance qui le sépare d'un point de vue donné. En effet, on représente par exemple un objet par des polygones, et plus on s'éloigne de cet objet, moins il y aura de polygones à traiter ce qui permettra un affichage plus rapide. Puisque l'on s'éloigne, les détails sont moins visibles, cela ne sert donc à rien de détailler de la même manière l'objet de près comme de loin. C'est pour cela que l'on a mis en place diverses types de LOD qui dynamiquement font varier le niveau de détail de l'objet.
Des algorithmes de simplification géométrique permettent de créer les LOD des objets pour les objets lointains, en mouvement rapide ou en périphérie du volume de vue.

1.2 Objectifs et contraintes

Selon la distance à laquelle est vu un objet en 3D, l'œil humain ne perçoit pas les détails de la même manière. Afin d'offrir une meilleure fluidité sans grande perte de détail, il y a possibilité pour le développeur de réduire le nombre de polygones en fonction de la distance.

- Ainsi, pour un objet éloigné, on créera un modèle peu précis.
- Pour un modèle plus proche, l'objet 3D sera un peu plus complexe et travaillé.
- Pour un vue de très près, l'ensemble des détails sera visible.

Cette évolution de la complexité se fait en temps réel. Il va donc falloir déterminer à quelle distance tel ou tel modèle appliquera un certain détail. D'autre part, la transition d'un modèle peu complexe à un modèle très complexe (et vice versa) doit être relativement discrète et perçue le moins possible à l'œil.

Dans notre projet, nous nous intéressons principalement à l'utilisation des niveaux de détails (LOD) pour la visualisation interactive de paysages urbains.

En visualisation 3D interactive de grandes zones urbaines, un certain nombre de problèmes se posent :

- le très grand volume de données à manipuler et à traiter,
- le « rendu » réaliste à réaliser,
- la vie à l'intérieur de la maquette,
- etc.

Nous allons donc explorer l'utilisation de certaines techniques pour la construction de LOD, principalement des techniques du type « opérateur global ».

L'objectif final est d'avancer dans la construction d'un modèle urbain utilisant des LOD qui offrira à la fois performance et qualité de rendu.

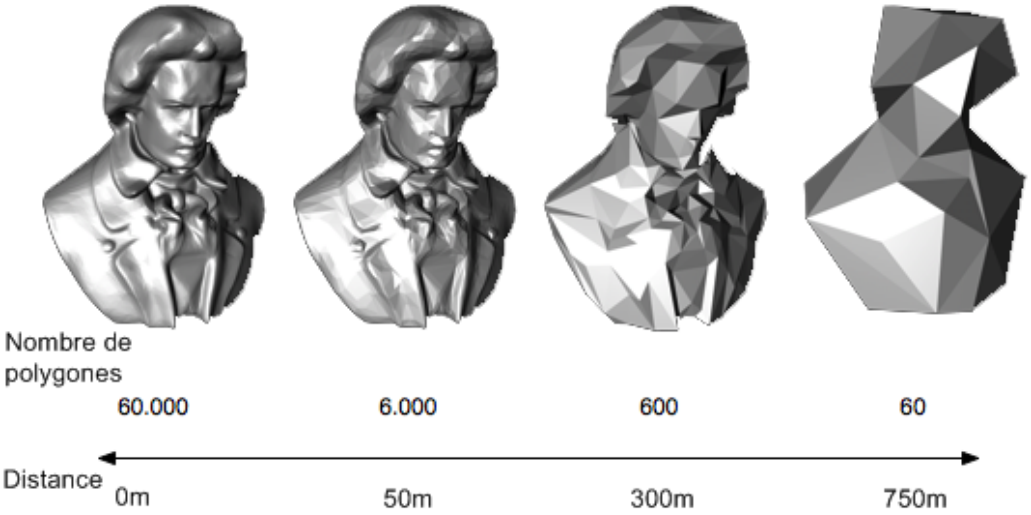
Dans ce contexte de modèle urbain, nous devrons appliquer et tester les différents algorithmes de génération de LOD. Au final cela nous permettra de réaliser une étude comparative de ces différents algorithmes.

2 Etat de l'art

2.1 Algorithmes des LOD

2.1.1 Les LOD discrets

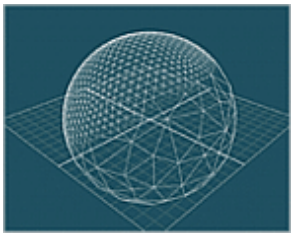
Les LOD discrets permettent de représenter le même modèle avec des versions de plusieurs niveaux de détails différents. Ces LOD sont classés dans la catégorie "LOD statiques" car ces versions ne sont pas générées en temps réel pendant la visualisation, mais avant celle-ci, et ne sont plus modifiées ensuite. Suivant la distance du modèle par rapport à l'observateur, on affichera le modèle avec le niveau de détail qui correspond. En effet, on charge tous les modèles qui représentent l'objet (il faudra donc prendre en compte le coût en mémoire), et on détermine celui qu'il faudra afficher. La liste des modèles est accompagnée d'une liste des distances correspondant à chaque modèle.
La figure ci-dessous montre les polygones d'un objet en fonction de la distance. Ces quatre modèles seront "chargés", et l'utilisateur verra, par exemple, le premier modèle devenir transparent par rapport au deuxième qui s'opacifiera afin d'éviter une vision de saccadement trop prononcée mais qui reste néanmoins visible si l'on y prête attention.



2.1.2 Les LOD continus

En continu, la visualisation est meilleure : le modèle se modifie lui-même en fonction du point de vue de la caméra. Contrairement aux LOD discrets, les changements ne seront presque pas perceptibles pour l'utilisateur. Un seul modèle est chargé, mais il est corrigé à chaque changement de position de la caméra, ce qui peut s'avérer plus coûteux en termes de performances, mais plus économe en mémoire du fait qu'il n'y a pas plusieurs instances du modèle qui sont stockées.
On peut également raffiner qu'une partie du modèle, qui correspondrait à la vision de

l'utilisateur. En effet, on accentuerait les détails de la partie visible de l'objet, tout en laissant peu de détails sur la partie cachée.

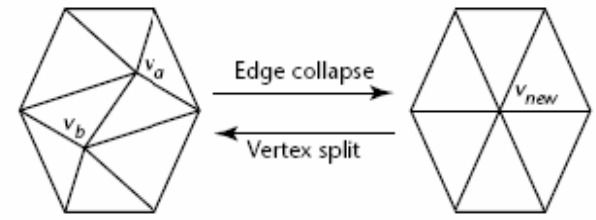


Exemple de la vision d'un utilisateur

2.2 Les techniques de simplification

2.2.1 Les opérateurs locaux

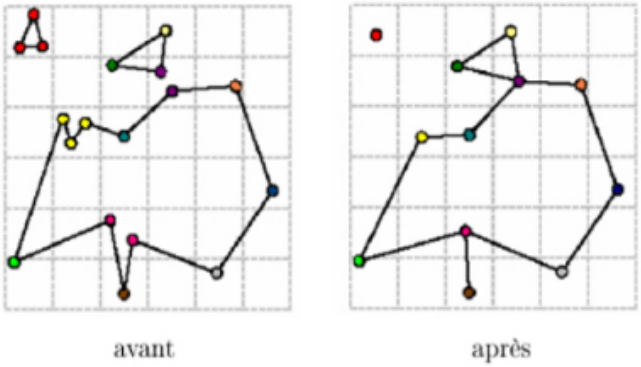
Les opérateurs locaux permettent de simplifier le maillage d'un objet de manière locale. Voici un exemple d'opérateur local : "l'edge collapse" proposé par Hoppe [Hoppe 96]. Cet opérateur transforme une arête (va, vb) en un sommet (vnew), il supprime l'arête ainsi que les triangles auxquels elle appartient. L'opération inverse est le "vertex split" qui additionne au maillage une arête et ses triangles adjacents (voir figure suivante). Cet opérateur permet de simplifier la géométrie avec le "edge collapse" et de la détailler avec le "vertex split".



Edge collapse de Hoppe : simplification et ajout de détail

2.2.2 Les opérateurs globaux

Les opérateurs globaux opèrent sur un très grand nombre de sommets en même temps (parfois, tout le maillage). Par exemple, le "cell collapse" permet de regrouper les sommets du maillage en un nombre fini d'ensembles. L'opération permet de transformer un ensemble de sommets, situés à l'intérieur d'un certain volume en un seul sommet. La cellule contenant les sommets peut appartenir à une grille ou à une subdivision spatiale telle qu'un octree [Rossignac & Borrel 93].



Cell collapse avec une grille régulière

Autre exemple de technique globale : les Surfaces Enveloppantes [Cohen et al. 96]
La simplification par surfaces enveloppantes positionne deux surfaces à l'intérieur et à l'extérieur du modèle à simplifier. Ces surfaces sont créées en déplaçant la surface originelle une certaine distance ε vers l'extérieur ou l'intérieur, le long des normales des sommets. La distance ε est spécifiée par l'utilisateur et peut être modifiée localement pour un sommet de manière à empêcher l'intersection des surfaces enveloppantes. La simplification est semblable à celle de la décimation de maillage : des sommets sont choisis pour être enlevés si à leur suppression le maillage simplifié reste dans l'espace défini par les deux enveloppes. Le principal avantage de cette méthode est la fidélité du maillage simplifié.

2.3 Quel LOD pour quelle situation ?

Le choix du LOD sera soumis à diverses contraintes : qualité de rendu, vitesse d'affichage, puissance de la machine ...
Comme vu précédemment, on peut simplifier en disant que les LOD statiques (discrets) offrent une moins bonne qualité de rendu que les LOD dynamiques (continus) ainsi qu'une plus grande consommation mémoire mais en contrepartie offrent de meilleures performances.

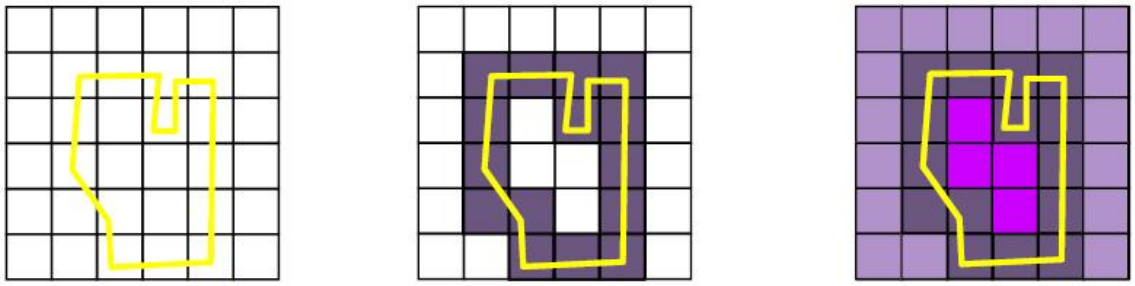
Dans les jeux vidéos, où il est ici question d'avoir de très bonnes performances (minimum 30 images/ seconde), les LOD statiques sont souvent privilégiés. En y prêtant même attention, on peut observer les transitions entre les modèles 3D comme vu précédemment dans l'explication du LOD discret.

(Ici, le jeux vidéo Crysis sur pc)

Par contre, dans les situations où la qualité graphique est privilégiée et que les performances sont secondaires, les LOD dynamiques seront préférés.

Une technique utilisé est le "Volume rendering".
On peut découper en plusieurs parties son fonctionnement :

- Rasteriser (transformer en voxels) le maillage dans une grille.
  Pour chaque facette du maillage, on va chercher les pixels 3D intersectés et les marquer comme "frontières". On va marquer les pixels 3D "extérieurs", puis marquer ceux restants comme "intérieurs".



- Filtrer passe-bas la grille pour éliminer les petits détails ou les trous
- Reconstruire un maillage à partir des voxel

3 Outils de développement

3.1 Langage de programmation

Pour la partie programmation, nous avons choisi le C++. Nous voulions un langage familier, mais surtout qu'il nous permette d'utiliser les librairies graphiques ainsi que les API nécessaires à notre projet tel que GLOD, QSlim, Ogre3D.

3.2 Outils et librairie : QSlim, Ogre3D, GLOD

3.2.1 Ogre3D

Ogre3D est un moteur 3D libre qui va nous permettre de créer facilement une scène contenant nos modèles 3D. Nous pourrons donc faire des tests et des comparatifs (de performance et de rendu) entre les différents algorithmes de LOD discrets que nous allons étudier et celui disponible dans Ogre lui même.

3.2.2 GLOD

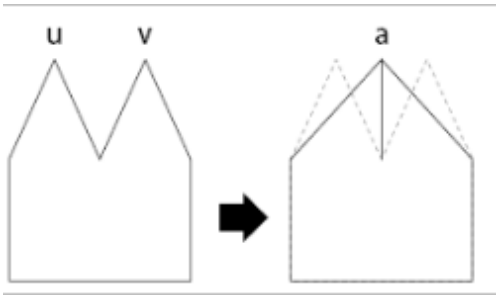
La bibliothèque GLOD permet de créer et de visualiser (par le biais d'OpenGL) des modèles 3D auxquels on applique des LOD discrets et continus, à la différence d'Ogre3D qui ne permet l'utilisation que de LOD discret. Nous pourrons ainsi faire des analyses et comparatifs des deux sortes de LOD.

3.2.3 Qslim

Qslim est un logiciel qui permet de simplifier le maillage des modèles 3D pour obtenir des représentations avec plus ou moins de triangles du même objet.

Un deuxième programme SMFView va nous permettre de visualiser les modèles .smf.

L'algorithme utilisé pour la simplification de maillage est l'algorithme QEM (Quadric Error Metrics) mis au point par Michael Garland et Paul Heckbert en 1997. [Garland 97]
Cette approche utilise un opérateur local d'effondrement de paires de points et une métrique d'erreur sommet-plans.



Le format fichier des modèles 3D utilisé par Qslim est le format SMF :
Le fichier utilise un maillage triangulaire, il est défini sous la forme suivante :

en début de fichier le mot begin

la liste des sommets, pour chaque sommet la lettre v suivie des 3 coordonnées

la liste des triangles, pour chaque triangle la lettre f suivie des indices des 3 sommets (indice à partir de 1)

en fin de fichier le mot end.

On génère ainsi d'autres modèles au format SMF avec des niveaux de détails différents.

Exemple de fichier SMF :

```
#$SMF 1.0
#$vertices 2904
#$faces 5804
#
# — The canonical cow —
#
# (data originally from powerflip, not avalon)
#
begin
v 0.151632 -0.043319 -0.08824
v 0.163424 -0.033934 -0.08411
v 0.163118 -0.053632 -0.080509
v 0.176307 -0.028912 -0.075048
v 0.174429 -0.051613 -0.073945
v 0.186153 -0.032952 -0.063704
v 0.189315 -0.049201 -0.055717
v 0.173804 -0.063906 -0.070661
v 0.161034 -0.067741 -0.076137
v 0.140389 -0.06919 -0.082181
v 0.145322 -0.055406 -0.086283
v 0.157816 -0.075509 -0.073519
v 0.138192 -0.077096 -0.078394
v 0.154266 -0.085543 -0.067386
v 0.135011 -0.089151 -0.072101
```

```
v 0.149841 -0.099483 -0.056737
v 0.171041 -0.083631 -0.062782
f 1 2 3
f 2 4 5
f 5 4 6
f 7 8 5
f 8 9 3
f 9 10 11
f 9 12 13
f 12 14 15
f 15 14 16
f 14 17 18
f 18 17 19
f 19 17 20
f 17 21 22
f 17 14 12
f 9 8 21
f 8 7 22
f 7 6 23
f 22 24 25
f 25 20 22
f 26 27 20
f 26 25 24

end
```

3.3 IDE

Pour l'environnement de développement logiciel, notre choix s'est porté sur CodeBlocks. Code :Blocks est un IDE multi-plate-forme publié sous licence GPL et développé en C++. Le principal avantage de CodeBlocks est sa portabilité, qui permet de reprendre ses projets de façon quasi-identique sous les différents systèmes supportés (Windows, Linux, Mac OS X) et avec les différents compilateurs (GCC, MSVC++, etc.). Il nous a aussi permis d'intégrer facilement les bibliothèques de Ogre3D et de GLOD. Il dispose des fonctionnalités classiques des IDE tels que interfaçage avec un débogueur (GDB), coloration syntaxique, complétion automatique du code etc.

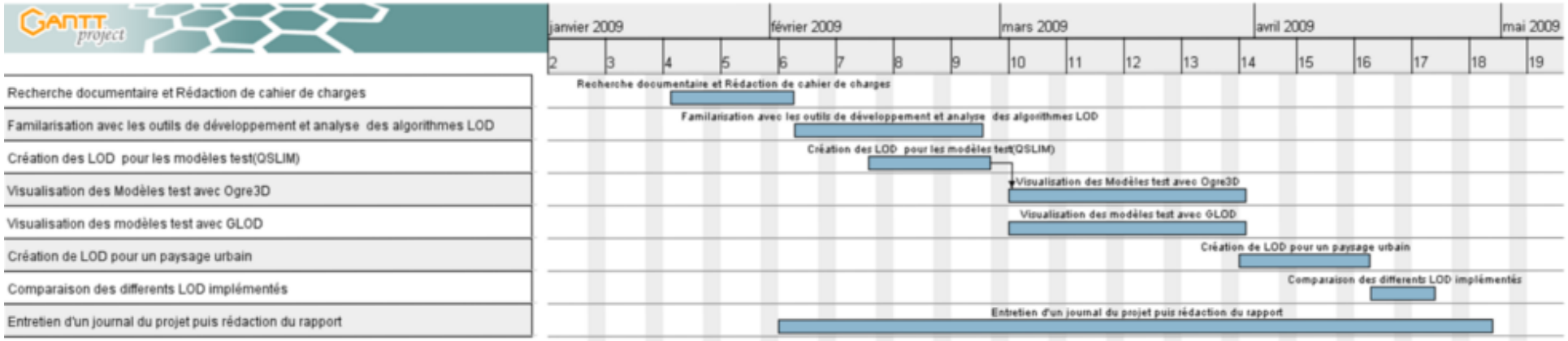
4 Organisation du travail

4.1 L'équipe

Nous avons réparti le travail en deux groupes de telle manière à ce qu'on puisse avancer dans notre projet en parallèle. En effet notre démarche est de diviser le projet en deux grandes phases :
Création de LOD discret pour des modèles test puis pour un paysage urbain à l'aide de QSLIM, visualisation avec la bibliothèque Ogre3D de modèles de tests et paysage urbain en utilisant les LOD créés qui seront traités par Gabriel et Mehdi.
Création de LOD discret et continu pour les modèles tests et paysage urbain et visualisation des ces modèles avec LOD en utilisant la bibliothèque GLOD, traité par Matthieu et Cécile.
Nous avons travaillé aussi individuellement pour les petites parties de codage, pour les recherches de documentation et pour la configuration des outils.
Au final, nous avons regroupé les résultats des deux parties pour une comparaison et une synthèse.

4.2 Le planning

Le planning prévisionnel établi dans le cahier des charges a bien été respecté, nous l'avions synthétisé dans le diagramme de Gantt suivant :


GANTT project
janvier 2009
février 2009
mars 2009
avril 2009
mai 2009
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
Recherche documentaire et Rédaction de cahier de charges
Familarisation avec les outils de développement et analyse des algorithmes LOD
Création des LOD pour les modèles test(QSLIM)
Visualisation des Modèles test avec Ogre3D
Visualisation des modèles test avec GLOD
Création de LOD pour un paysage urbain
Comparaison des differents LOD implémentés
Entretien d'un journal du projet puis rédaction du rapport

4.3 La démarche

4.3.1 Deux projets parallèles

Ogre3D

Cette partie consistait à créer des LOD discret pour les modèles test puis pour des modèles de paysage urbains à l'aide de la bibliothèque QSLIM et de les visualiser avec le moteur 3D Ogre3D
Notre partie s'est déroulée en plusieurs étapes :

1. Création de LOD discret pour les modèles tests avec la bibliothèque QSLIM.
2. Création d'une scène avec ogre3D avec gestion des caméras pour le déplacement et visualisation des modèles de base intégrés à Ogre3D.
3. Chargement des modèles test avec LOD générés sous QSLIM dans cette même scène, ainsi que l'utilisation des LOD de Ogre3D sur les modèles précédents.
4. Création d'une scène avec intégration des modèles urbains, en utilisant les LOD de QSLIM et Ogre3D.

GLOD

Cette partie consistait à écrire un programme en C / OpenGL / GLOD qui nous permettrait de tester le rendu et les performances des LOD discret et continu sur différents types de modèles. La bibliothèque GLOD (Geometric Level Of Detail : niveau de détail géométrique) permet de visualiser des modèles 3D et d'appliquer des LOD discrets et continus.
La programmation s'est déroulée en trois grandes étapes :

1. Chargement de fichiers PLY (cf 5.2.2) à l'écran et modification du niveau de détail en appuyant sur une touche. Nous avions chargé quelques modèles et, à l'aide du clavier, nous pouvions voir les détails apparaitre ou disparaître.
2. Création d'une vraie scène avec déplacement, et modification du niveau de détail suivant la distance, et de manière indépendante pour chaque objet. Nous avons, pour ce faire, intégré plusieurs modèles de géométrie différentes.
3. Création d'une même scène (un quartier) visualisé également en Ogre3D afin d'effectuer des comparatifs entre les deux projets (cf 4.3.3)

4.3.2 Les modèles utilisés (Test)

Tout d'abord, nous allons commencer par comparer les performances sur quelques objets (par exemple des modèles de lapins, de vaches et de chevaux).

Le fait de tester nos LOD sur différents types d'objets nous permet de voir la qualité de rendu des LOD (si on reconnaît encore bien l'objet après déformation) et leurs performances. On espère ainsi pouvoir en déduire quel type de LOD convient le mieux à tel type de scène/ objet.

4.3.3 Création du paysage urbain

La création du paysage urbain consistait à créer une ville commune aux deux projets, avec le même nombre de bâtiments, avec une certaine orientation. Le but est de comparer les performances de Ogre3D avec celles de GLOD et voir quelle est la solution la mieux adaptée à une ”navigation dans un paysage urbain”.

5 Présentation des deux projets parallèles

5.1 Ogre3D

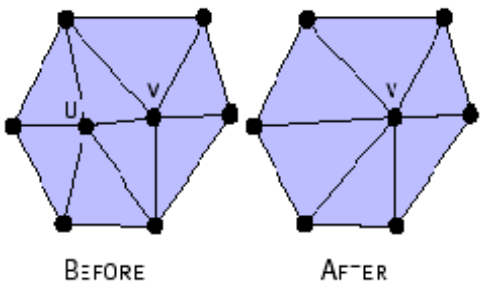
5.1.1 Présentation

Nous allons utiliser le moteur 3D Ogre3D [Ogre3D] afin de visualiser des modèles et de pouvoir appliquer différents LOD. Ogre3D donne la possibilité d'appliquer un algorithme pour générer les différents niveaux de détails des objets que l'on va utiliser, cet algorithme Progressive Mesh type Polygon Reduction Algorithm by Stan Melax [Melax 98] reprend des aspects de la méthode de Hoppe. Seule la topologie est modifiée : déplacement d'une extrémité sur l'autre.

Ogre3D possède deux manières de calculer le nombre de sommets qui sont réduits à chaque LOD :

VRQ_PROPORTIONAL : Une proportion du nombre de sommets restant est enlevée à chaque réduction.

VRQ_CONSTANT : Un nombre défini de sommets est enlevé à chaque réduction.



Progressive Mesh type Polygon Reduction Algorithm by Stan Melax

L'utilisation de cette technique permet un temps de calcul moins couteux car on tient compte de la géométrie :

- Distance entre les deux extrémités de l'arête
- Angle maximal entre deux faces voisines (permet de conserver les angles)

Dans un premier temps, nous allons utiliser des modèles simples d'animaux, afin de tester les différentes techniques de LOD. Nous allons nous intéresser au LOD que nous propose

Ogre3D puis nous essaierons d'utiliser un logiciel externe QSLIM [QSlim] à notre application qui va nous générer le modèle que l'on souhaite avec différents niveaux de détail choisis par nos soins.
Puis, nous allons essayer d'appliquer les LOD sur des bâtiments pour créer un paysage urbain, ce sera la deuxième partie.

5.1.2 Les modèles

Formats des modèles

- **Format Mesh :**
  Il existe de nombreux formats pour les modèles 3D, il n'y a pas de standard imposé, donc beaucoup de formats se sont développés au fil du temps. Ogre3D utilise le format Mesh (un format propre à ce moteur 3D).
  Le fichier .mesh contient l'ensemble des points et triangles qui constituent le modèle3D. Un fichier .Material est associé au .mesh et contient les liens vers les textures (fichiers images) et les shaders.
  On peut lui associer un fichier .skelton qui défini les animations sur l'objet.

- **Format SMF :**
  QSlim utilise le format SMF (très peu répandu), donc il va falloir utiliser des outils de conversions [Outils conversion] afin de passer du format .mesh au format .smf et inversement, car une fois que l'on aura généré avec QSlim nos modèles avec différents LOD, il faudra les reconvertir en .mesh pour pouvoir les utiliser dans Ogre3D. Nous allons utiliser le format 3DS afin de passer d'un format à un autre.

- **Format 3DS :**
  Contrairement au format Mesh, pour le format 3DS, il existe de nombreuses ressources disponibles sur internet pour trouver des modèles. Ce format utilisé dans le logiciel 3D Studio Max (Logiciel de modélisation et d'animation 3D) est assez répandu.

- **Format sketch :**
  Utilisé par le logiciel de modélisation 3D Google SketchUp, ce format permet d'avoir accès à beaucoup de modèles de bâtiments dont nous avons besoin pour réaliser la partie paysage urbain. Le format Sketch sera ainsi converti en 3DS puis en .mesh si on utilise directement le modèle dans Ogre3D ou il sera converti en .smf si on souhaite générer ce modèle avec d'autres niveaux de détails.

Outils utilisés

- **Google Sketchtup :**
  Google SketchUp est un programme complet qui permet de modéliser en 3D des bâtiments et toutes sortes d'objets édités par le partenariat des Sociétés de Google

et @lastSoftware.
Dans notre projet, il nous sert à modéliser et récupérer des modèles 3D de bâtiments.

- **3ds2mesh**
  Cet outil permet de convertir le format 3DS vers le format de Ogre3D (.mesh/.material)

- **IVCON 3D Graphics File Converter**
  Converti le format .smf au format .3ds

Les modèles utilisés

Les modèles test : Pour tester les différents LOD de Qslim et Ogre3D, nous avons utilisé des modèles simples d'animaux.

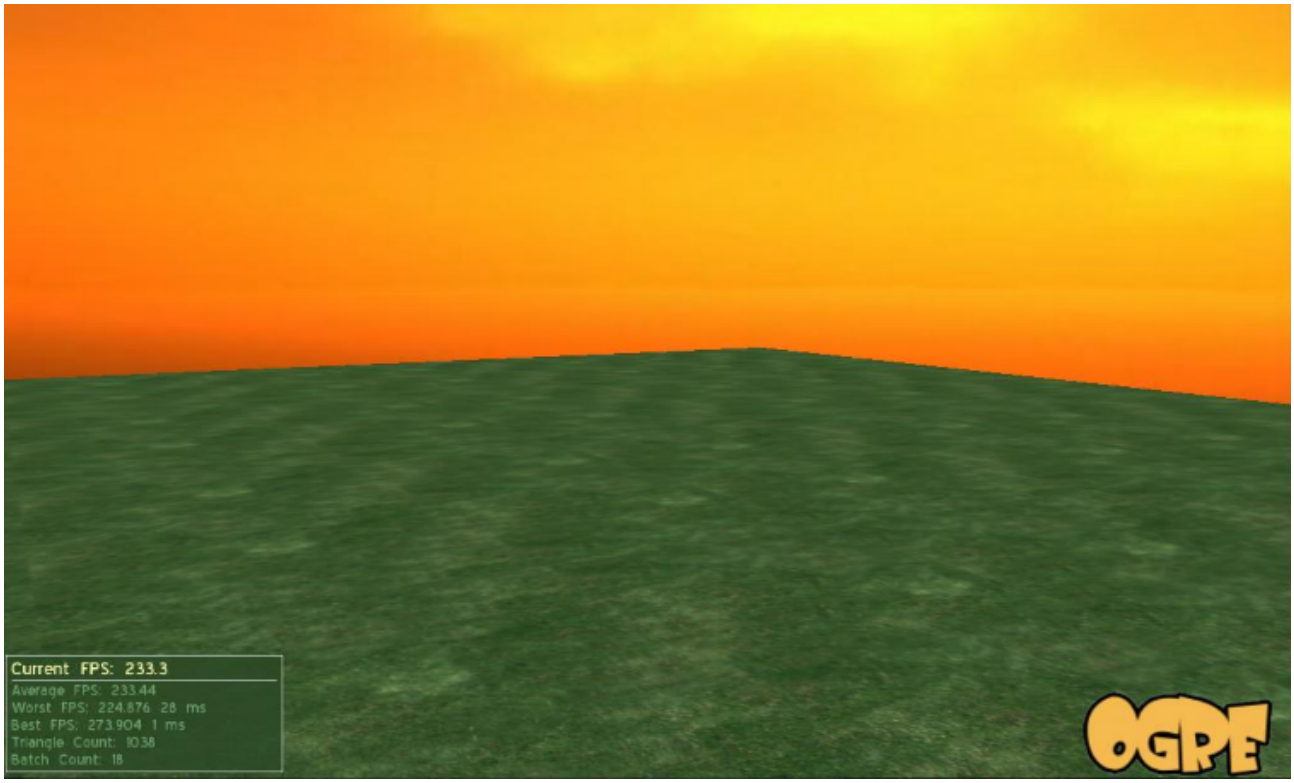
Les modèles des paysages urbains : Afin de créer un paysage urbain, nous avons utilisé quatre types de modèles de bâtiments différents.

5.1.3 La programmation avec Ogre3D

Nous avons donc réalisé un programme permettant de charger plusieurs modèles simultanément afin de pouvoir représenter une scène composée de plusieurs modèles 3D dont chacun aura son niveau de détail qui sera modifié selon la distance de l'utilisateur.
Les différentes étapes de la programmation sont les suivantes :

Réalisation de la scène

On va hériter directement de la classe abstraite ExampleApplication qui va nous permettre de créer une première application, elle contient le canevas par défaut d'une classe d'application pour le moteur Ogre. On va disposer de la variable mSceneMgr qui contient l'objet courant de la classe SceneManager, créé par ExampleApplication.
C'est le SceneManager qui gère tout ce qui apparaît à l'écran tels que la caméra, la lumière, mais aussi le placement de tous les objets de la scène.
Nous allons directement créer une caméra avec le SceneManager puis on règlera sa position et sa direction. On pourra également gérer la distance de coupure (la distance à laquelle on ne verra plus l'objet) afin de faciliter la vue à travers les objets à l'écran quand on est très proches d'eux.



Scène vide sous Ogre3D

Intégration des modèles

L'intégration d'un modèle va se faire en plusieurs étapes. Tout d'abord, il est nécessaire de charger le fichier .mesh. Une fois notre modèle chargé, on va pouvoir gérer les LOD (cette partie est expliquée dans le prochain point).
Ensuite, nous devrons créer une entité à partir du SceneManager en lui donnant un nom unique et on lui associe le fichier .mesh que l'on vient de charger. Une entité est, en fait, un objet que vous pouvez rendre dans une scène.

*Entity * ent1 = mSceneMgr->createEntity("vache ", "cow.mesh");*

Maintenant que nous avons notre entité dans le SceneManager, on va créer un objet SceneNode qui est un nœud dans le SceneManager avec un nom unique et une position puis on va attacher notre entité à ce nœud. Le SceneNode contient les informations au sujet du positionnement et de l'orientation des entités qui y sont rattachées.

*SceneNode * node1 = mSceneMgr->getRootSceneNode()->createChildSceneNode("CowNode");*

node1->attachObject(ent1);

Scène avec modèle sous Ogre3D

Gestion des modèles

Sous Ogre3D, il est possible de générer différents LOD suivant un vecteur de distance. Une fois l'objet Mesh chargé on va créer un vecteur qui va contenir les distances des différents LOD à afficher.

```
MeshPtr mesh2 = MeshManager : :getSingleton().load("horse.mesh",
ResourceGroupManager : :DEFAULT_RESOURCE_GROUP_NAME) ;
    Mesh : :LodDistanceList dists ;
    dists.push_back(128) ;
    dists.push_back(256) ;
    dists.push_back(512) ;
    dists.push_back(1024) ;
    mesh2->generateLodLevels(dists, ProgressiveMesh : :VRQ_PROPORTIONAL, Real(0.5)) ;
```

Ogre3D permet aussi au développeur de charger ses propres LOD. Ainsi, on peut rajouter des LOD à l'objet que l'on a chargé tout en choisissant les distances auxquelles on veut afficher les LOD.

```
MeshPtr mesh1 = MeshManager : :getSingleton().load("cow.mesh", ResourceGroupMana-
ger : :DEFAULT_RESOURCE_GROUP_NAME) ;
```

mesh1->createManualLodLevel(128, "cow-1000.mesh");
mesh1->createManualLodLevel(256, "cow-200.mesh");

5.1.4 Contraintes et difficultés

L'installation de Ogre3D n'a pas été simple, nous avons utilisé CodeBlocks comme plateforme de développement. Bien qu'Ogre3D s'est bien intégré, il a fallu lier notre projet aux librairies de Ogre3D à plusieurs endroits.
Le manque d'homogénéité des formats de modèles 3D a été aussi un problème au début de notre projet. Il nous a contraint à gérer les conversions d'un format à un autre pour pouvoir utiliser correctement les modèles.
La prise en main de Ogre3D s'est faite assez naturellement, grâce aux nombreuses documentations disponibles en français. Cependant, il a fallu approfondir les recherches afin de maîtriser le fonctionnement de certaines méthodes de gestion de LOD.

5.1.5 Conclusion

Nous n'avons pas constaté de grandes différences entre le rendu visuel des LOD discret de QSlim et ceux de Ogre3D. Nous avons constaté aussi un gros avantage à utiliser nos propre LOD générés au préalable avec Qslim .
En effet, cela nous donne la possibilité de choisir les niveaux de détails de l'objet et la distance à laquelle on va les afficher ; tandis qu'avec les LOD de Ogre3D, on ne pourra choisir que la distance à laquelle il souhaite changer les niveaux de détails sans pouvoir les préciser. Si on veut maîtriser le rendu de ses modèles ou optimiser la performance, il est plus intéressant de créer soi-même les LOD de son modèle et de les intégrer par la suite dans Ogre3D.
On pourra noter qu'avec les LOD discrets, les modèles sont chargés ou créés au lancement de l'application (et non en temps réel) ce qui permet d'optimiser les calculs durant la visualisation et d'augmenter la fluidité.

Nous avons constaté aussi que les LOD discrets ne sont pas adapatés sur des modèles urbains tels que les maisons ou les bâtiments de part leur forme anguleuse dont les normales des facettes forment souvent des angles obtus.

5.2 GLOD

5.2.1 Présentation

Comme nous venons de le voir, OGRE 3D ne permet l'utilisation que de LOD discret. La bibliothèque GLOD [GLOD] (Geometric Level Of Detail) permet de créer et de visualiser (par le biais d'OpenGL [OpenGL]) des modèles de LOD discrets et continus dans une API légère, flexible, de haute performance. Ainsi, les LOD sont générés rapidement et peuvent être affichés dans une scène.
Cet API permet d'avoir des LOD applicables à toutes sortes de modèles 3D.

5.2.2 Le format des modèles : PLY

Afin d'utiliser GLOD, il nous est nécessaire d'utiliser des modèles 3D. Nous avons choisi le format de fichier PLY [PLY], car c'est un format léger qui suffit à nos expérimentations, nos modèles ne contenant qu'une liste de sommets et de faces. Nous l'avons bien étudié car nous n'en avons pas seulement réalisé l'importation, mais également l'exportation.

C'est un format pour le stockage des objets graphiques qui sont décrits comme un ensemble de polygones. L'objectif est de fournir un format qui est simple et facile à mettre en œuvre mais qui est suffisamment général pour l'utilisation d'un grand nombre de modèles.

Le format PLY décrit un objet comme une collection de sommets, de faces et d'autres éléments optionnels, avec des propriétés telles que la couleur et la direction. Les propriétés peuvent être stockées avec l'objet contenant la couleur, les coordonnées de textures, la transparence, et différentes propriétés de l'avant et l'arrière d'un polygone.

Structure d'un fichier PLY basique :
Entête
Liste de sommets
Liste de faces
(Listes d'autres éléments)

L'entête comprend une description de chaque type d'lément avec le nom de l'élément (par exemple "point"), le nombre d'élément présent dans l'objet et une liste des différentes propriétés associées à l'élément. L'entête indique également si le fichier est binaire ou ASCII.

5.2.3 La programmation en GLOD

La programmation avec GLOD est un peu particulière mais permet beaucoup de choses.

Pour appliquer un LOD à un objet 3D, nous importons sa liste d'arêtes et de sommets à un objet GLOD en décidant si nous voulons y appliquer un LOD continu ou discret.

Création d'un objet GLOD :

// Création d'un groupe d'objet Glod glodNewGroup(NumGroup) ;
// Création d'un objet Glod que l'on affecte au groupe d'objet
glodNewObject(NumObjet,NumGroup,ModeLOD) ; // ModeLOD = Discret ou Continu
Nous avons donc la possibilité d'associer plusieurs objets GLOD à un groupe d'objet ou de les laisser indépendants les uns des autres. Ceci permet d'appliquer notre LOD sur un groupe d'objet ou sur des objets individuels (pour des résultats différents). Pour finir nous pouvons appliquer un taux « d'erreur » en temps réel sur l'objet, plus l'erreur est grande et plus il sera déformé ce qui correspond à la déformation du LOD.

glodGroupParameterf(NumGroup, GLOD_OBJECT_SPACE_ERROR_THRESHOLD, taux_d'erreur) ;
En LOD discret, différentes instances du modèles seront crées à l'exécution et chacune correspondra à une déformation du modèle. Par exemple à 10 mètres, on affichera un modèle moins déformé qu'à 100 mètres. On se contente donc d'afficher tel ou tel instance selon la distance. C'est en ça que le LOD discret est très différent du LOD continu. Le continu supprime des sommets/arrêtes aux modèles selon la distance du point de vue et restructure en conséquence l'objet. On imagine la différence en terme de calcul.

5.2.4 Le logiciel

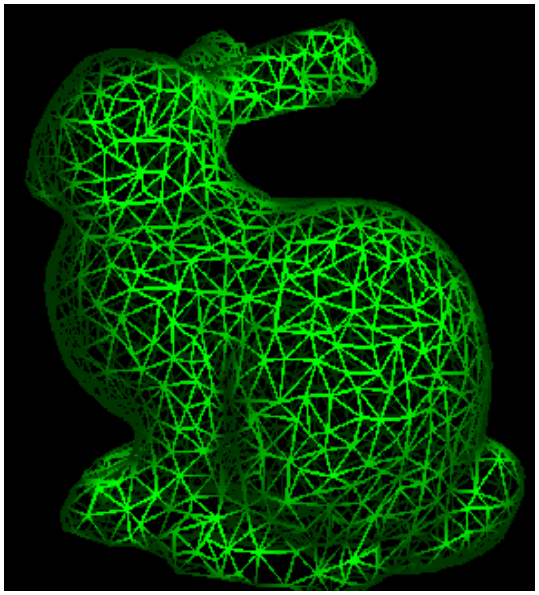
Nous devions donc réaliser un programme permettant de charger plusieurs modèles simultanément afin de pouvoir représenter une scène composée de plusieurs modèles 3D dont chacun aura son niveau de détail qui sera modifié selon la distance de l'utilisateur.

Pour ce faire, nous avons créé un fichier scène qui contient les chemins des différents modèles 3D de la scène avec leurs positions, leurs orientations, leurs numéros de groupes d'objet ainsi que le type de LOD (discret ou continu), ce qui permet d'afficher une scène contenant différents type de LOD et d'objets.

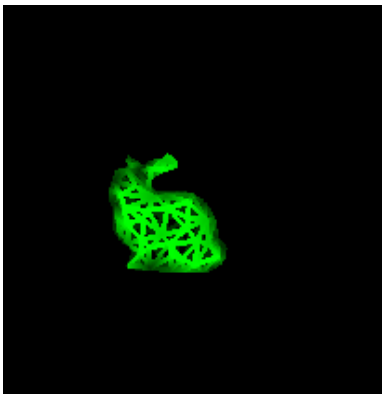
Au lancement du logiciel, une interface nous permet de saisir le nom du fichier scène à importer :

Une fois ceci fait, le logiciel charge tous les modèles et nous permet de les visualiser dans la scène 3D et de nous y déplacer (ce qui fera varier leurs niveaux de détails).

Exemple de rendu (en fil de fer) suivant la distance en continu :



Distance : 14,26 (78fps)



Distance : 19,68 (217fps)

Nous voyons donc par cet exemple que nous pouvons nous déplacer dans notre scène afin de voir en temps réel l'impact du LOD sur l'objet (qui est fonction de la distance, de la position du point de vue). Nous voyons également que le LOD appliqué garde la géométrie de l'objet et que même avec moins de polygones, on reconnaît notre lapin.

A chaque déplacement, notre distance avec les différents objets est prise en compte et appliquera en fonction le degré d' « erreur » sur chaque objet suivant son LOD spécifié.

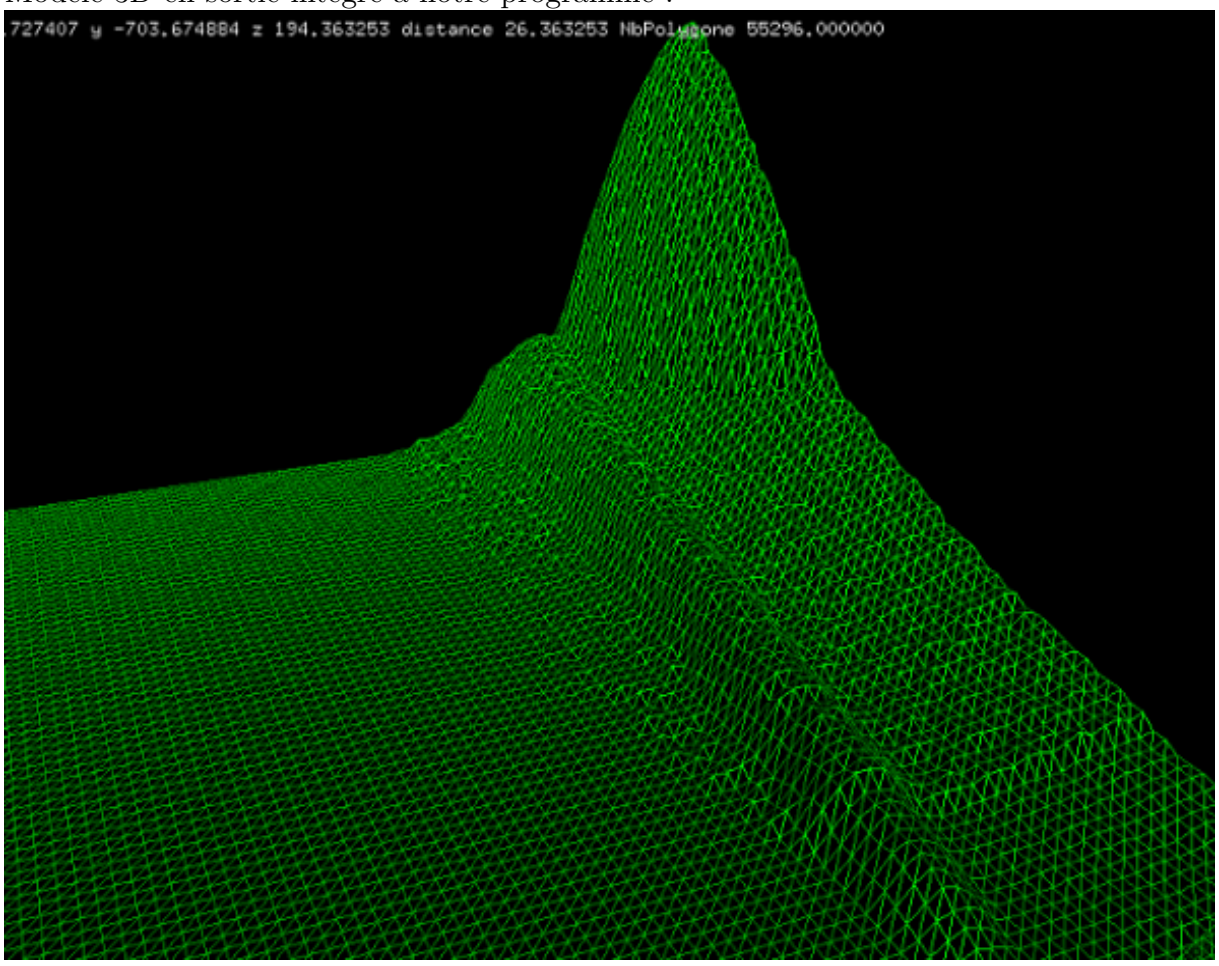
Remarque : En LOD continu, modifier les modèles consomme beaucoup plus de ressources du fait de la restructuration qui est inexistante en LOD discret.

Le logiciel nous affiche le nombre d'images/seconde en temps réel.

Nous voulions également pouvoir expérimenter les performances des LOD sur des objets de types « terrain ». Pour cela nous avons programmé la conversion d'un fichier image, contenant différents niveaux de gris, en un fichier .ply. Chaque niveau de gris correspondant à une hauteur, noir est le point le plus bas et blanc le point le plus élevé.

Fichier image de données :

Modèle 3D en sortie intégré à notre programme :



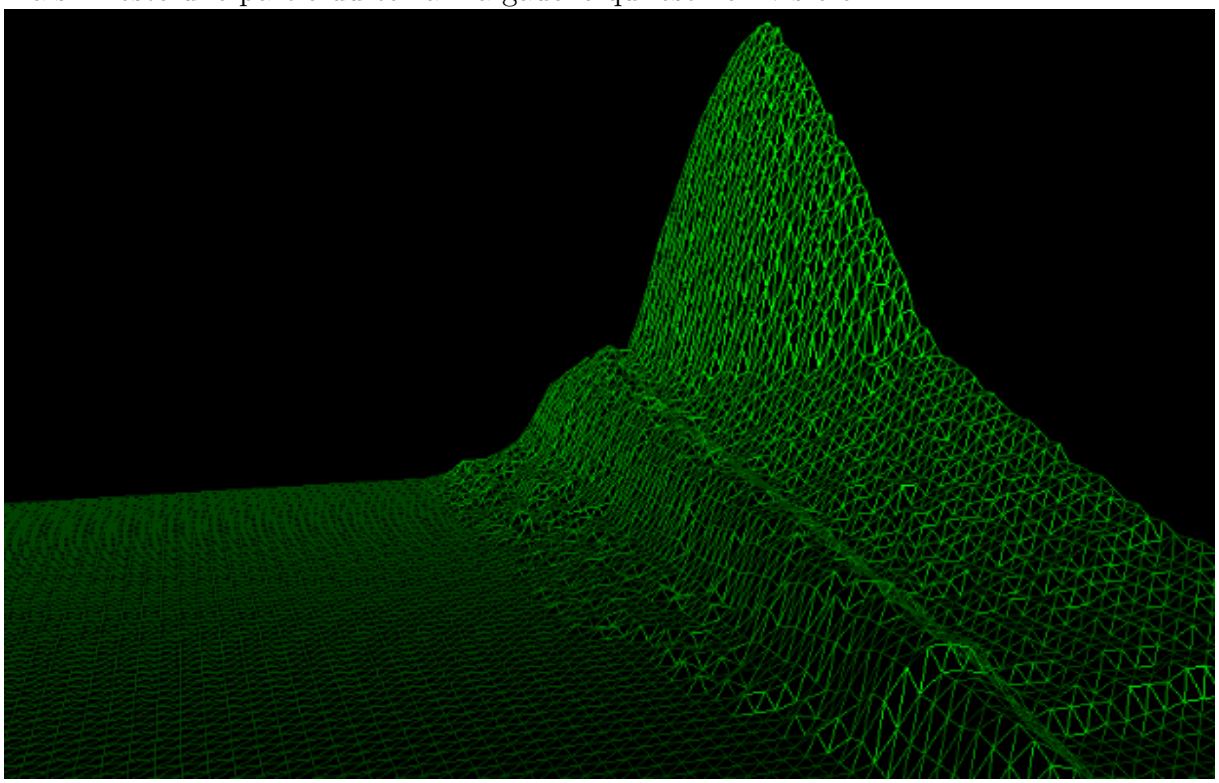
Pour réaliser cette conversion, nous lisons chaque pixel de l'image, sa position dans l'image est la position du sommet dans la scène et sa couleur est sa hauteur. Ensuite à partir de ces informations, nous créons un fichier .ply contenant la liste de ces sommets et triangles afin de l'importer facilement dans notre logiciel.

Nous avons donc pu tester les performances des deux types de LOD également avec un terrain.

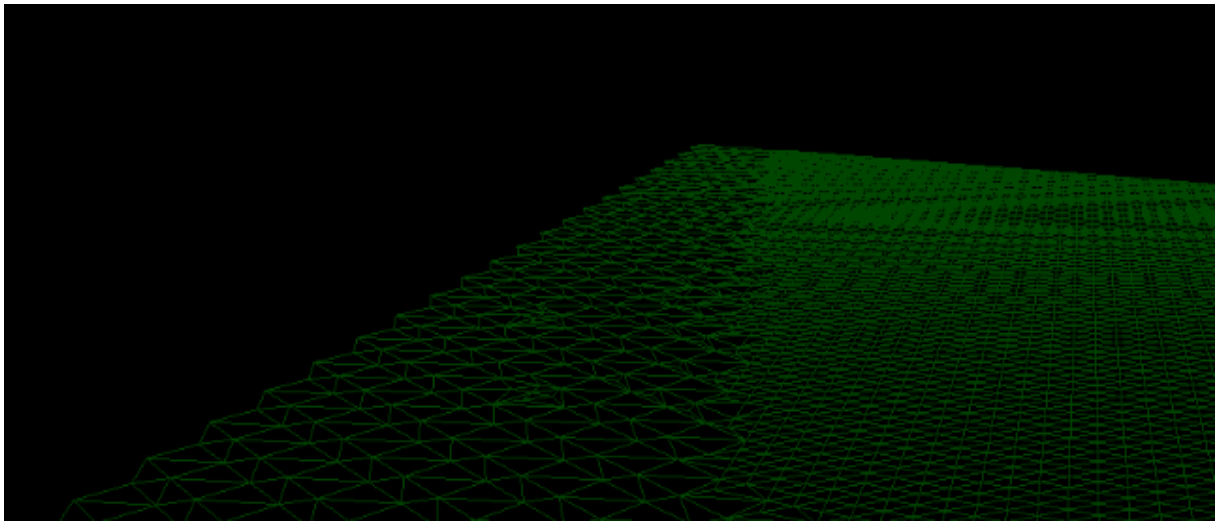
Tester nos LOD sur un objet 3D de type "terrain" nous a permis de comprendre l'importance du fait que le LOD continu est dépendant du point de vue à la différence du LOD discret.
Cela se traduit par le fait qu'en LOD continu, juste la partie de l'objet qui est visible à l'écran est détaillé alors qu'en LOD discret, même si l'on ne voit qu'une infime partie de l'objet, son ensemble sera détaillé.
Nous pouvons donc en déduire que les LOD de type continu sont à préférés en cas de terrain.

Lorsque le point de vue change en LOD continu, nous pouvons voir (du fait de la "lenteur" de la restructuration), ce qui était invisible à l'instant d'avant, se reconstruire au moment où il devient visible. Ce qui permet donc de bien voir l'apport de la dépendance du point de vue.

Exemple :
Nous avons la colline dans notre champ de vision ainsi qu'un morceau de terrain à gauche, mais il reste une partie du terrain à gauche qui est non visible :



Si l'on regarde à gauche nous voyons que le sol est en train de se construire car il devient visible :



Alors qu'en LOD discret, l'objet entier resterait construit, qu'on le voit entièrement ou non.

5.2.5 Difficultés rencontrées

Au cours de notre projet, la principale difficulté aura été le manque de documentation, d'exemple et de l'inexistence d'une communauté sur internet concernant la bibliothèque GLOD.
La maitrise d'OpenGL nous aura également demandé un temps d'adaptation ainsi que celle des outils de conversion.

5.2.6 Conclusion de la partie GLOD

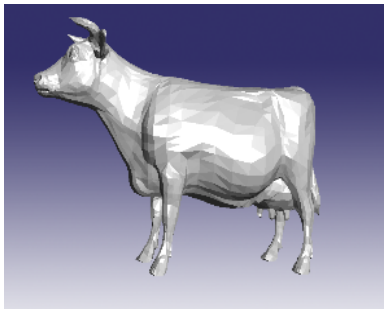
GLOD nous aura permis de comparer les LOD continus et discrets pour une scène donnée. L'intérêt du LOD continu est évident pour de grands objets (par exemple un terrain), car seul ce qui est visible par l'utilisateur est affiché contrairement au LOD discret où tout l'objet (le terrain) serait affiché. On pourrait penser alors que l'algorithme de LOD continu est à choisir dans tous les cas... et bien nos tests nous ont prouvés que non. La perte de détails de l'objet est beaucoup plus douce qu'en LOD discret mais le rendu est alors beaucoup plus lent car le processeur doit déterminer ce qui est visible par l'utilisateur et restructurer continuellement les modèles 3D. Ce qui signifie qu'on enlève du travail à la carte graphique (moins de polygones à afficher) mais on en rajoute au processeur (restructuration de l'objet). Ce n'est pas forcément un bon choix pour des objets autres que des terrains car il y a un tel écart entre les performances des cartes graphiques et des processeurs que l'on perd en performance.

Nous comprenons alors pourquoi le LOD discret est souvent préféré pour une question de performance car tous les modèles sont déjà présents en mémoire et nécessitent peu de calculs de la part du processeur.

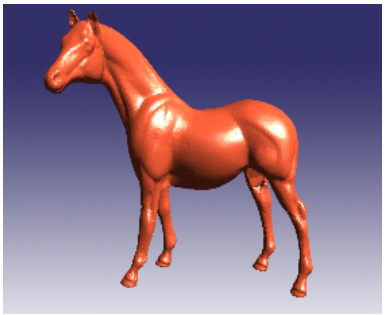
6 Synthèse et comparaison des résultats des deux équipes

6.1 Présentation des objets de comparaison (test animaux/ test Urbains)

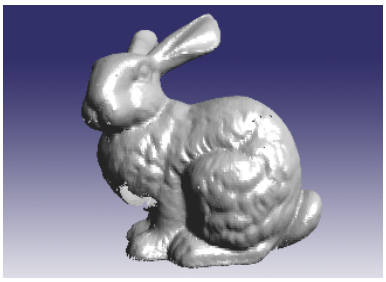
Nous avons cherché à modéliser les mêmes scènes, d'une part avec quelques animaux, et de l'autre un paysage urbain afin de comparer les deux possibilités de rendu.
Tout d'abord, dans la scène modélisant les animaux, nous avons choisi comme modèles une vache, un lapin et un cheval, que nous avons représenté un certain nombre de fois.



Une vache [vache]

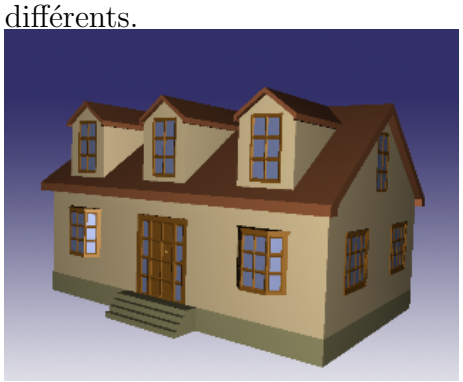


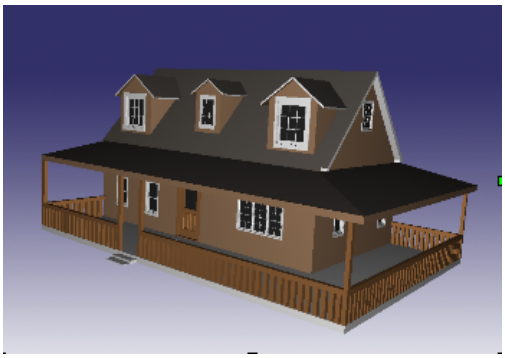
Un cheval [cheval]

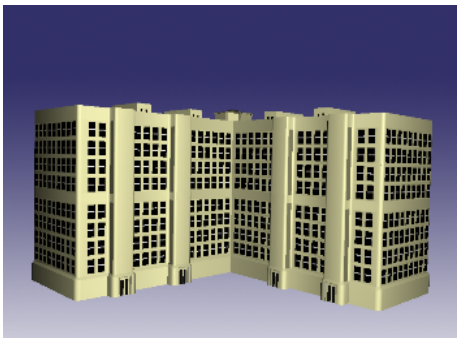


Un lapin [lapin]

La deuxième scène représente deux types de maisons [maisons] et de buildings [buidings] différents.



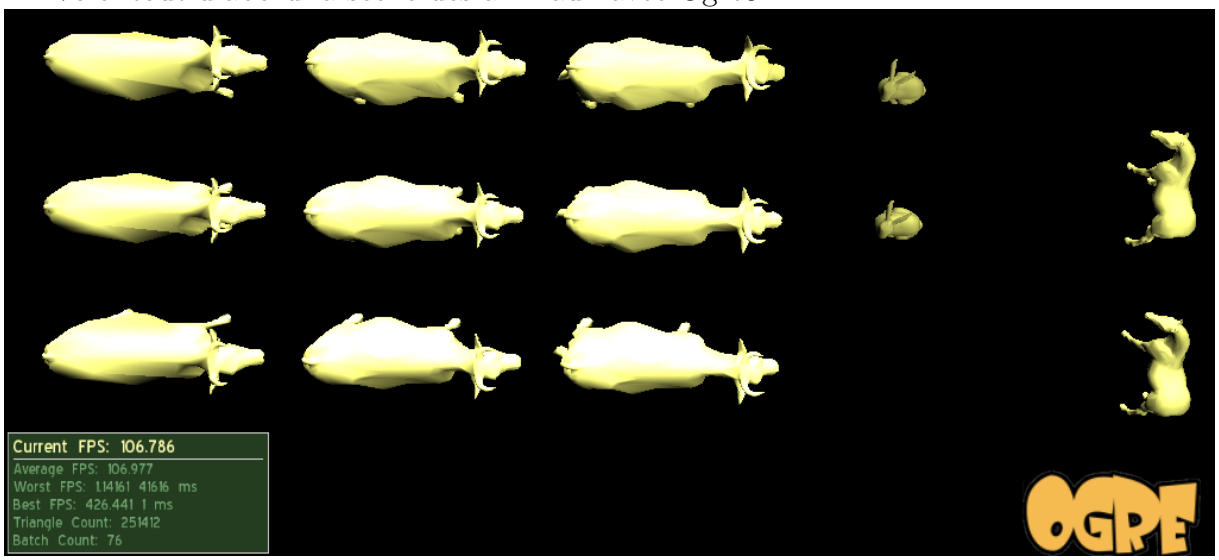




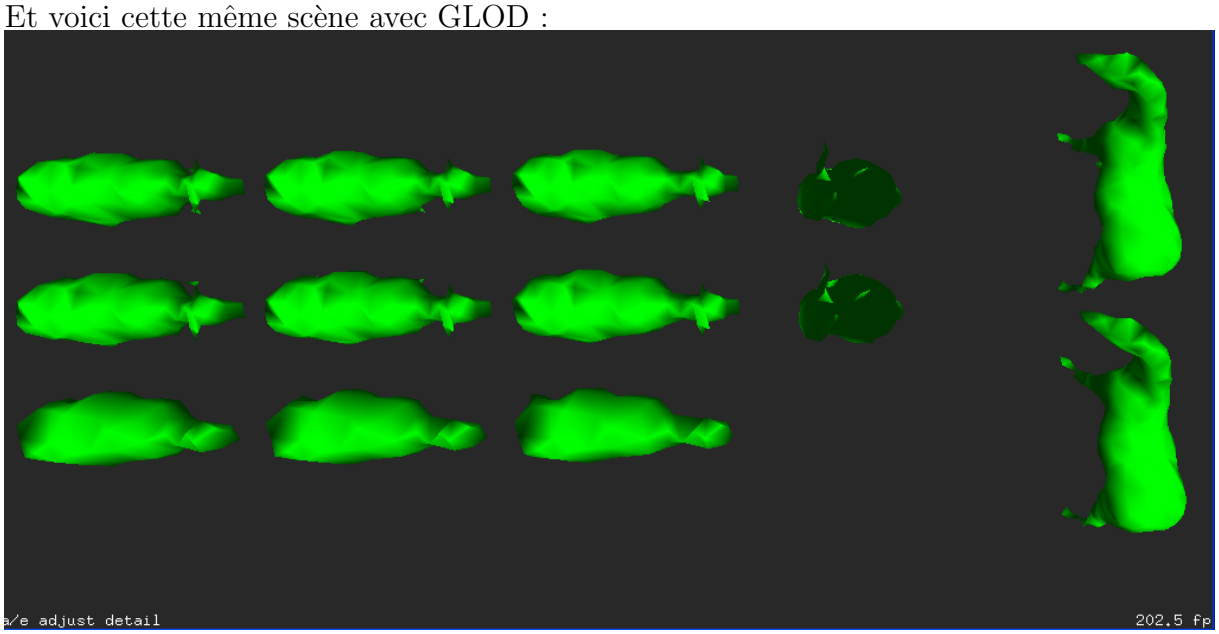
6.2 Mêmes modèles avec Ogre3D et GLOD

6.2.1 Scène de la ferme

Voici tout d'abord la scène des animaux avec Ogre3D :



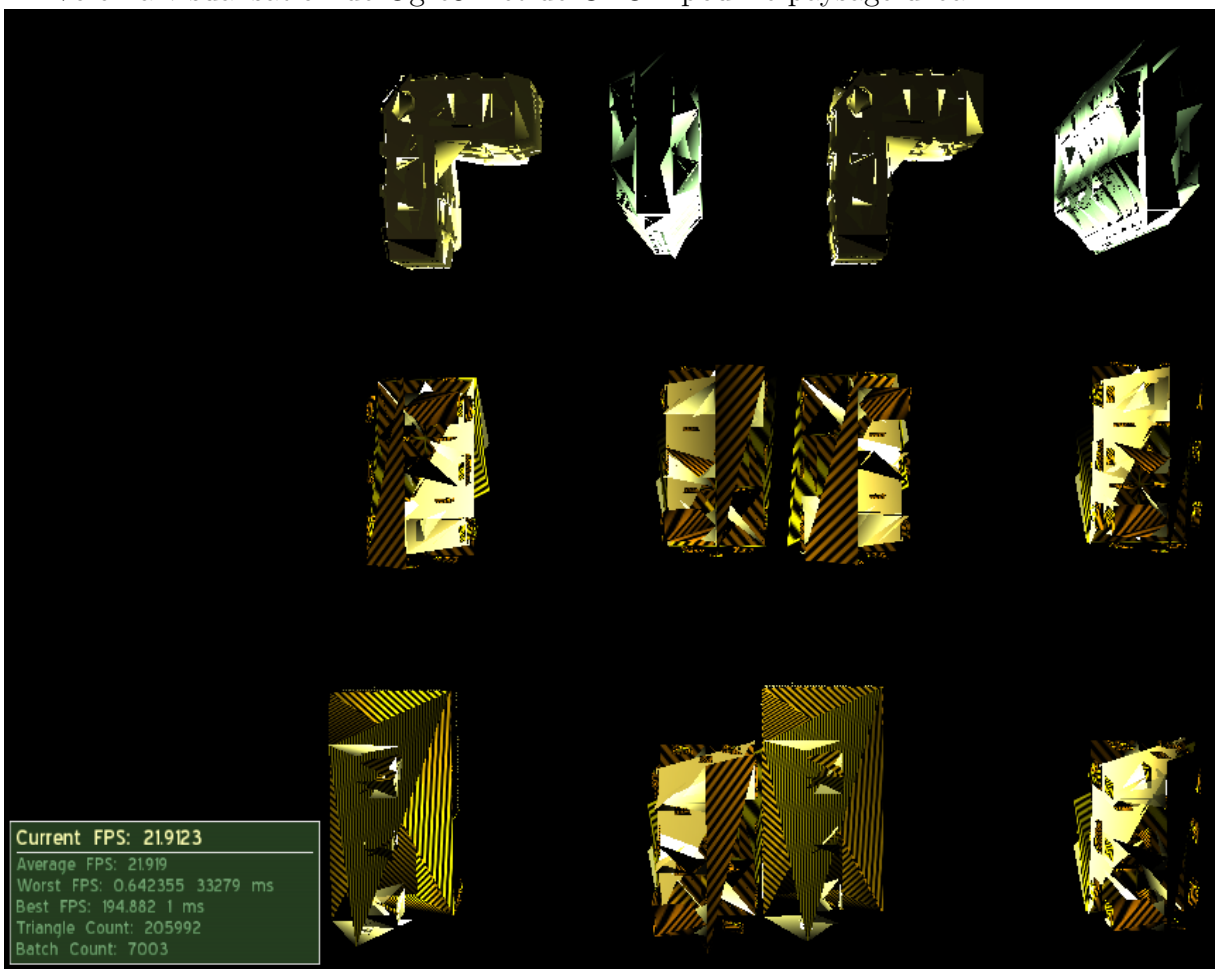
Et voici cette même scène avec GLOD :

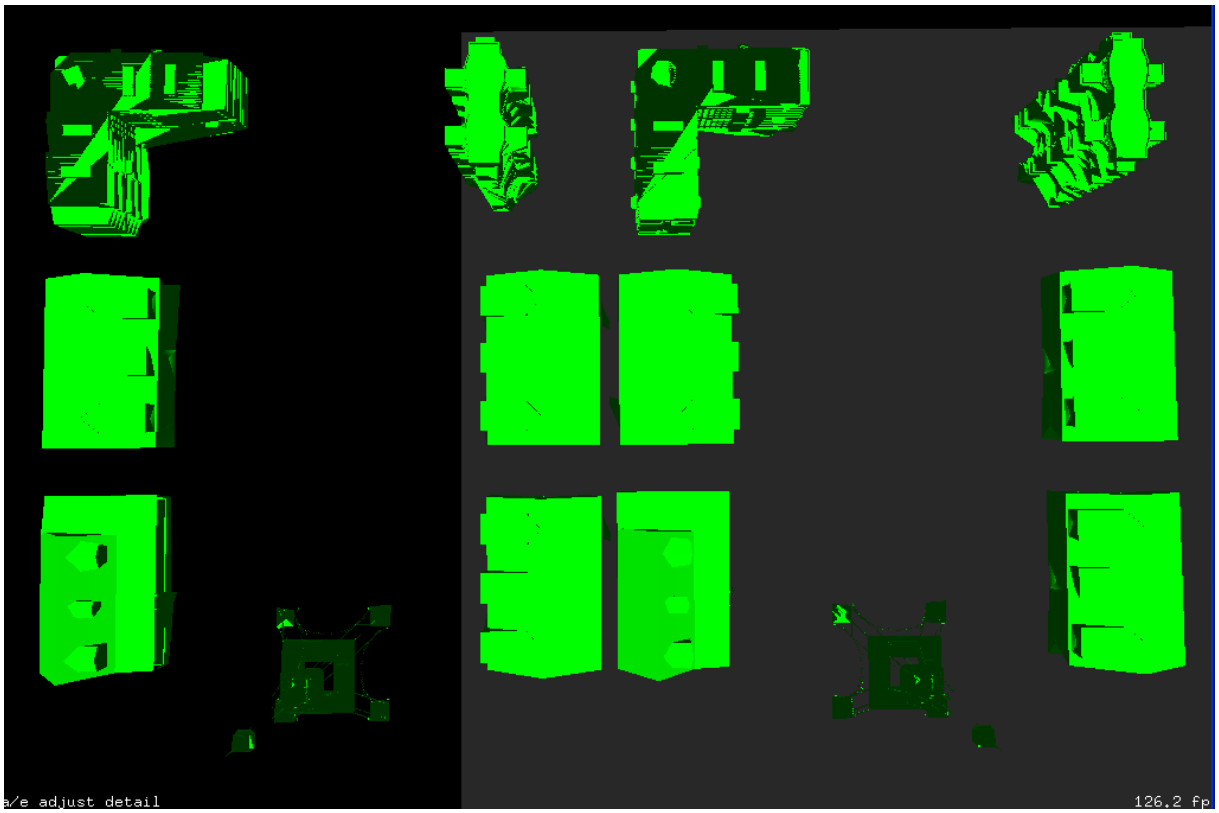


- Nombre d'objets : Pour cette scène, nous avons au total 13 objets.
- Nombre de polygones : 166 356 polygones
- FrameRate
  - Pour Ogre3D : environ 106,7 FPS
  - Pour GLOD : environ 202 FPS
- Qualité visuelle : Les détails sont plus approfondis en Ogre3D

6.2.2 Paysage urbain

Voici la visualisation de Ogre3D et de GLOD pour le paysage urbain :





- Nombre d'objets :
  - Pour Ogre3D : 12 objets
  - Pour GLOD : 14 objets

- Nombre de polygones : 349 270 polygones

- FrameRate
  - Pour Ogre3D : environ 21,9 FPS
  - Pour GLOD : environ 126 FPS

- Qualité visuelle : Rendu meilleur en GLOD

6.3 Tableaux de comparaison

		GLOD			Ogre3D	
	Distance	Sans LOD	LOD discret	LOD continu	Sans LOD	LOD discret
Ferme : 166 356 polygones	5	FPS : 47 Rendu : 5/5	FPS : 186 Rendu : 4/5	FPS : 46 Rendu : 2/5	FPS : 41 Rendu : 5/5	FPS : 72 Rendu : 5/5
	10	FPS : 47 Rendu : 5/5	FPS : 221 Rendu : 2/5	FPS : 82 Rendu : 2/5	FPS : 43 Rendu : 5/5	FPS : 114 Rendu : 5/5
	20	FPS : 47 Rendu : 5/5	FPS : 249 Rendu : 1/5	FPS : 122 Rendu : 0/5	FPS : 44 Rendu : 5/5	FPS : 215 Rendu : 4/5
Ville : 349 270 polygones	20	FPS : 23 Rendu : 5/5	FPS : 86 Rendu : 5/5	FPS : 1,1 Rendu : 3/5	FPS : 64 Rendu : 5/5	FPS : 45 Rendu : 5/5
	30	FPS : 23 Rendu : 5/5	FPS : 111 Rendu : 4/5	FPS : 1,4 Rendu : 2/5	FPS : 22 Rendu : 5/5	FPS : 35 Rendu : 3/5
	50	FPS : 23 Rendu : 5/5	FPS : 192 Rendu : 1/5	FPS : 2 Rendu : 1/5	FPS : 22 Rendu : 5/5	FPS : 22 Rendu : 1/5

Pour Ogre3D tout comme pour GLOD, nous avons constaté que les performances sont moindres lorsque l'on effectue un affichage en fil de fer (on fait afficher les arrêtes). Cela peut s'expliquer par le fait que GLOD ou Ogre3D doit afficher toutes les faces du modèle que l'on ne voit pas lorsque l'objet est opaque. Mais nous n'avons pas utilisé un tel mode d'affichage pour nos comparaisons.
Pour le même paysage urbain, nous constatons que GLOD est plus performant que Ogre3D (126FPS pour GLOD contre 21,9FPS pour Ogre3D).

6.4 Conclusion (Les algo utilisés sont-ils adaptés à la modélisation des paysages urbains ?)

A la vue de ces comparaisons, nous voudrions en déduire qu'une solution est meilleure que l'autre mais ce n'est pas aussi simple. Car même si l'affichage en LOD discret avec GLOD est plus rapide, la solution d'Ogre3D n'est pas à écarter car elle offre une meilleure qualité d'affichage.

Alors que choisir ? Et bien cela dépend du type de scène à afficher et du rendu voulu (performance/ qualité d'affichage).
Nos comparaisons nous ont montré que GLOD est globalement plus performant en LOD discret que Ogre3D, au détriment de la qualité d'affichage.
De plus, Ogre3D est plus facile à utiliser du fait de sa programmation par objet (en C pour GLOD), et nécessitera beaucoup moins de lignes de code pour représenter la même scène qu'en GLOD et ses performances seront suffisantes pour une scène pas trop chargée.

En conclusion, on peut dire que chacune de ces deux solutions permettent de modéliser des paysages urbains, en LOD discret, avec néanmoins des réserves quant à Ogre3D en cas de zone trop chargée.

7 Bibliographie

[Hoppe 96] Hugues Hoppe. Progressive meshes. Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pages 99-108. ACM Press, 1996.

[Rossignac & Borrel 93] J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering. B. Falcidieno and T. L. Kunii, editors, Modeling in Computer Graphics, pages 455-465. Springer-Verlag, June-July 1993.

[Cohen et al., 96] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, P. Frederick Brooks, Jr., and William Wrigh. Simplification envelopes. Proceedings of the ACM Conference on Computer Graphics, pages 119-128, New Orleans, LA, USA, August 1996. ACM

[Garland 97] Michael Garland and Paul Heckbert. Surface Simplification Using Quadric Error Metrics. Proceedings of SIGGRAPH 97.

[Melax 98] Stan Melax : A simple, fast and effective polygon reduction algorithm. Game Developer Magazine, November 1998.
http ://www.melax.com/gdmag.pdf

Les modèles :
[vache] http ://mgarland.org/dist/qslim-1_0.zip

[cheval] et [lapin] http ://ljk.imag.fr/membres/Nicolas.Szafran/ENSEIGNEMENT/MASTER2/VISU/TP6/

[maisons] http ://sketchup.google.com/3dwarehouse/ ?hl=fr

[buildings] http ://www.marlinstudios.com/samples/sampbldg.htm

[GLOD] http ://www.cs.jhu.edu/ graphics/GLOD/
http ://www.cs.virginia.edu/ luebke/publications/pdf/GLOD.techreport.pdf

[OpenGL] http ://nehe.gamedev.net/

[PLY] http ://fr.wikipedia.org/wiki/PLY
http ://local.wasp.uwa.edu.au/ pbourke/dataformats/ply/

[Ogre3D] http ://ogre3d.fr/wiki/Accueil
http ://ogre3d.fr/forums.html
http ://gusgus.developpez.com/Ogre/
http ://helios.univ-reims.fr/Labos/LERI/membre/bittar/Ogre3D/td01.html
http ://www.joserodolfo.com/mesh_lod_with_ogre

[Qslim] http ://mgarland.org/software/qslim.html
http ://ljk.imag.fr/membres/Nicolas.Szafran/ENSEIGNEMENT/MASTER2/VISU/TP6/

[Outils conversion] http ://people.sc.fsu.edu/ burkardt/f_src/ivread/ivread.html http ://david.geldreich.free.fr/dev.html

Outils de visualisation :

http ://www.glc-player.net/fr/index.php
http ://mgarland.org/class/model/SMFView.exe