2015

Plaquage d’un modèle 3d sur MNT

MOHAMMED AMIRI

HATIM LAMGHARI

ZAKARIA AIT OMAR

SOMMAIRE :

I. INTRODUCTION

Dans le cadre des cours dispensés dans le master TSI, il nous a été demandé d'effectuer un projet de plaquage d'un modèle 3D sur un modèle numérique de terrain (MNT), ce travail peut être un socle pour les simulations de construction de bâtiments et leur faisabilité dans l'espace. Ce projet une fois réalisé entièrement peut constituer un outil pour le suivi de l'évolution d'un ensemble de bâtiments et peut être conjugué à d'autres projets pour identifier des zones à risque (exemple d'inondation).

II. CONCEPTION ET GESTION DE PROJET

a. *Analyse du sujet:*

La tâche principale du projet est de pouvoir plaquer un objet 3D simple sur le MNT, pour ce faire nous avons procéder à une analyse exhaustive pour définir les différentes fonctionnalités de l'application. Nous avons découpé le projet comme suit :

- Une interface Homme-Machine : nous avons décidé de créer trois fenêtres de visualisation "**ViewerMnt**", "**ViewerObjet** " et "**ViewerObjetFinal**" respectivement pour afficher l'objet plaqué sur le MNT, l'objet initial et l'objet plaqué.
- Lecture et affichage du MNT et application d'une texture : cette composante de l'application permet de lire le fichier XYZ, et de l'afficher à l'écran une fois texturé.
- Chargement et affichage de l'objet 3D : cette fonctionnalité permet de lire et afficher le modèle 3D objet du plaquage.
- Plaquage du modèle 3D sur le MNT : cette fonctionnalité consiste à identifier les points d'intersection de l'objet avec la surface superficielle du MNT et de le tracer suivant le relief.

L'ensemble des classes qui ont découlé de l'analyse approfondie du sujet sont données en détail dans le diagramme de classe joint dans l'annexe n°1. Nous nous limitons ici aux classes principales:

- La classe **MNT :** répond au besoin de lecture et d'affichage moyennant trois méthodes
- La classe **Objet :** permet de lire et afficher l'objet
- La classe **plaquage :** englobe les différents algorithmes permettant de manipuler l'objet en vue de son plaquage.

Afin de réaliser les différentes tâches, nous nous sommes servis des librairies suivantes :

- Pour le rendu 3D : **OpenGl**
- Pour la manipulation de la vue : **QGLViewer**
- Pour charger l'objet 3D : **Assimp**

b. *Planning et méthodes de projet :*

L'organisation du projet a été faite suivant les méthodes agiles en particulier "**SCRUM**". Après avoir listé les grandes tâches (user story), elles ont été subdivisées en tâches élémentaires au sein de sept **sprint**, puis classées par ordre de priorité, elles ont été affectées des poids correspondant à leur difficulté estimés selon la suite de fibonacci, le tableau récapitulatif des sprints et la figure de l'évolution temporelle du travail illustrant la gestion du projet sont fournis ci-après.

	Tâche	Poids	théorique	Réalisation	pratique
Sprint 1	recherche et prise en main de Qt	89	733	75	733
	diagramme de classe	55	681	40	658
Sprint 2	interface graphique initiale	21	629	21	618
	Lecture du fichier XYZ	55	577	55	563
Sprint 3	Triangulation	34	525	34	529
	Affichage des points	34	473	34	495
Sprint 4	Classe DEM (MNT)	21	421	15	480
	Texture du MNT	34	369	26	454
Sprint 5	Affichage Modèle 3D	89	317	60	394
	Exemple de plaquage	55	265	30	364
Sprint 6	Réglage de la scène	34	213	34	330
	plaquage d'un modèle 3D	89	161	80	250
Sprint 7	algorithme de plaquage	89	109	70	180
	finalisation et rapport	34	57	34	100
		733	0		

Figure 1 Tableau de Sprint

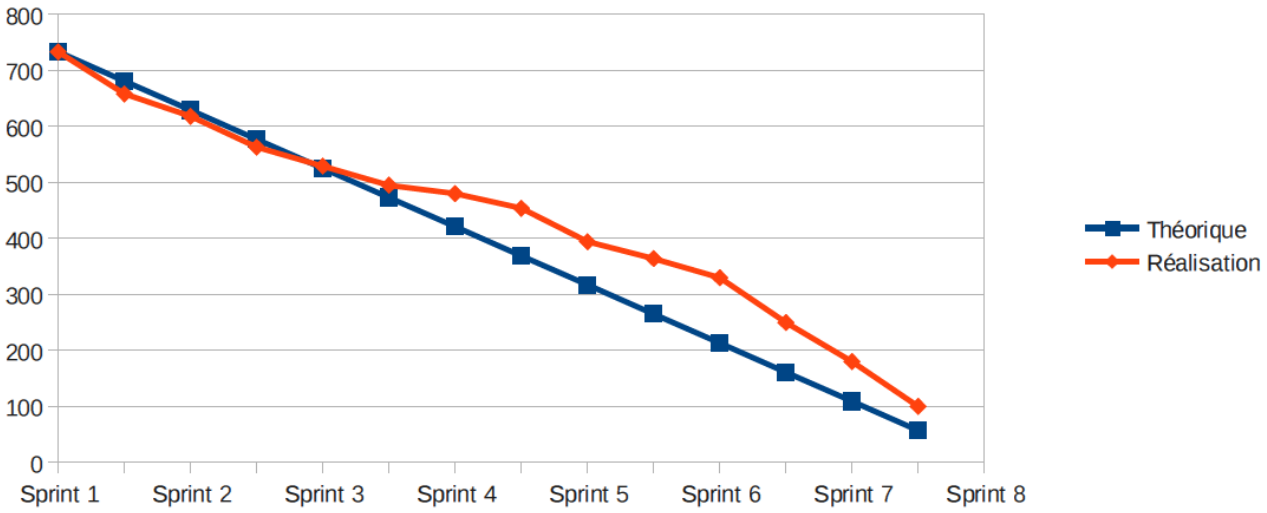

Figure 2 : Burndown Chart

III. DEVELOPPEMENT :

a. Chargement et affichage du MNT :

Algorithme de chargement et de lecture du MNT :

Les coordonnées des points constituants l'MNT sont ordonnées suivant trois colonnes de gauche à droite contenant respectivement l'abscisse l'ordonnée et l'altitude. Afin de charger ces données nous avons défini trois variables **x**, **y**, et **z** de type **float** pour les coordonnées plus un tableau **Qvector<QVector3D>** pour les stocker. Pour l'ouverture du "fichier.xyz" nous avons utilisé le flux de lecture "**flux**" instancié à l'aide du constructeur **QTextStream,** ensuite pour la lecture des variables l'opérateur **>>** , les variables sont stockées immédiatement dans un vecteur **QVector3D** qui lui est inséré dans le tableau à l'aide de la fonction **push_back().** L'ensemble de ses étapes et performé à l'intérieur d'une boucle **While** ayant comme condition "**not end of file".** Deux compteurs **i** et **colonne** ont été placés dans la boucle ayant pour rôle respectivement de dénombrer les points constituant la maille de l'MNT et le nombre de points suivant la direction Y. Le nombre de points suivant X "**ligne**" est obtenu par division de **i** par **colonne.** Le nombre d'arrêtes suivant une direction est le nombre de points diminué de 1. Nous avons déclaré alors quartes variables supplémentaires pour contenir ces différents paramètres.

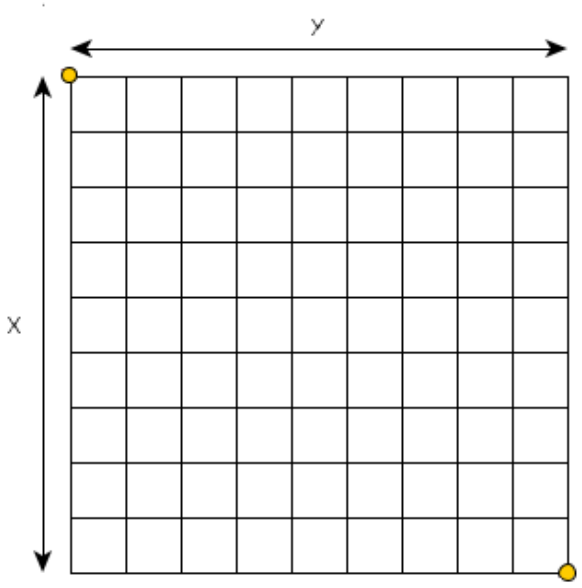

Figure 3 Maille de l'MNT

Affichage de l'MNT :

Pour dessiner l'MNT nous avons utilisé la bibliothèque **OpenGL**, pour une visualisation aisée nous avons utilisé la librairie **QGLViewer**.

Initialisation de la vue:

Les paramètres initiaux de la fenêtre **Viewer** contenant l'MNT sont insérés dans la méthode **init()**,

les fonctions **setBoundingBox(point min, point max)** et **showEntireScene()** servent indiquer à la caméra le volume à visualiser autour de l'emprise de l'MNT.

Algorithme du dessin de l'MNT:

La bibliothèque **OpenGL** propose plusieurs primitives pour réaliser un rendu 3D, pour des besoins de texture nous avons retenu la primitive **GL_TRIANGLES.**

Il existe plusieurs méthodes pour envoyer des données à la carte graphique en vue de leur dessin, on pourra envoyer les points constituant la grille de l'MNT de différentes manières, par ordre de performance croissant : envoyer chaque point un par un, envoyer un tableau de points, un tableau d'indices ou encore les stocker dans un buffer. Nous avons testé toutes les méthodes cependant nous avons retenu la deuxième qui présente un compromis entre les performances du rendu 3D et les exigences en terme de machine.

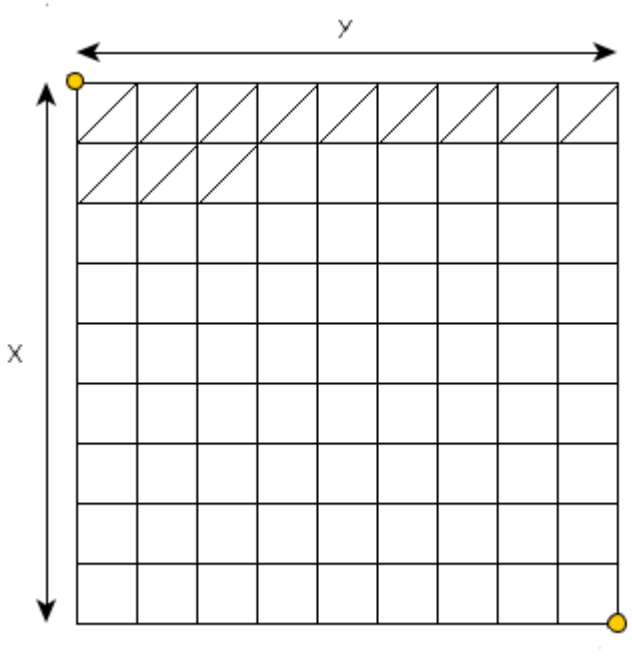

Figure 4 Triangulation de la maille de l'MNT

Afin de dessiner la maille de l'MNT, nous allons créer un tableau à partir du tableau initial rempli lors du chargement que l'on va remplir de manière à ce que chaque carré est dessiné moyennant deux triangles. L'indice d'un point dans la maille est donné par la somme de sa colonne et de sa ligne multipliée par le nombre de colonne. Pour reproduire un carré il faut donc insérer les point d'indices

i , i+1 et i+nombre de points suivant Y pour le premier triangle, ensuite les points d'indices i+1, i+points suivant Y et i+1+points suivant Y. Pour représenter l'intégralité des carrés il faut faire deux boucles imbriquées de 0 à nombre d'arrêtes suivant X et de 0 à nombre d'arrêtes suivant Y.

Nous avons maintenant notre tableau contenant des points ordonnés de façon à dessiner des triangles, il faut donc l'envoyer au processeur graphique, pour ce faire on informe notre carte graphique que nous allons utiliser un tableau à l'aide de la fonction **glEnableClientState(GL_VERTEX_ARRAY),** on lui indique notre tableau à l'aide de la fonction **glVertexPointer** qui prend trois paramètres : le nombre de composantes dans un vertex (trois dans notre cas), le type de données (double) et un pointeur constant vers le tableau. Le dessin est réalisé par la fonction **glDrawArrays()** qui prend comme paramètres le type de primitive (GL_TRIANGLES) et le nombre de point. Le dessin de l'MNT se déroule à proprement dit au sein de la fonction **draw()** de notre **Viewer.**

Application d'une texture :

La texture retenue pour notre MNT a été réalisé a partir du fichier .xyz à l'aide d'une interpolation de couleur avant d'avoir l'idée d'utiliser une photo Land-Sat d'extension .PNG.

La fonction **LoadTexture** s'occupe de charger la texture en mémoire et de lui créer un identifiant.

Pour appliquer cette texture il fallait faire correspondre à chaque point de notre grille un point de la texture, nous avons donc crée un tableau pour les coordonnées de la texture. Les coordonnées de la texture sont par défaut compris entre 0 et 1 (voir figure suivante).

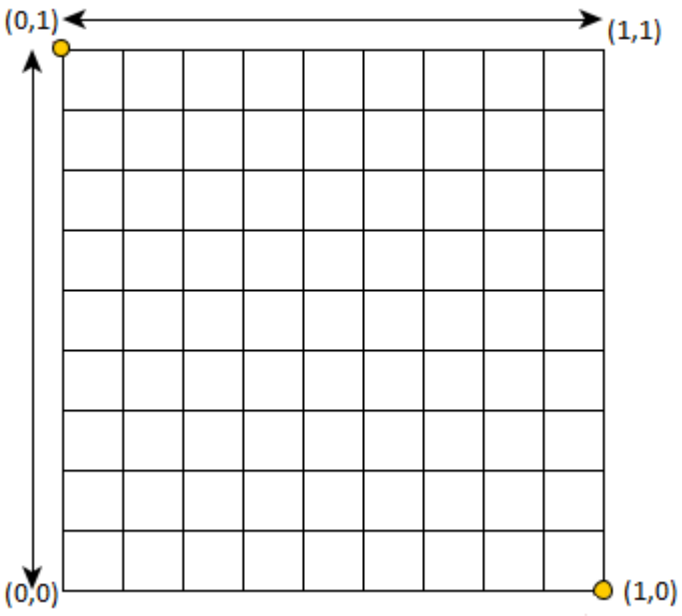

Figure 5 Coordonnées du MNT

Les coordonnées d'un point de la grille de la texture sont construit en divisant les indices de la ligne et de la colonne respectivement par le nombre d'arrêtes suivant X et le nombre d'arrêtes suivant Y.

La texture est activé dans la fonction **draw()** à l'aide de la fonction **glBindTexture()**. Lors du rendu, le tableau de la texture est envoyé à la carte graphique à l'aide de la fonction **glTexCoordPointer().**

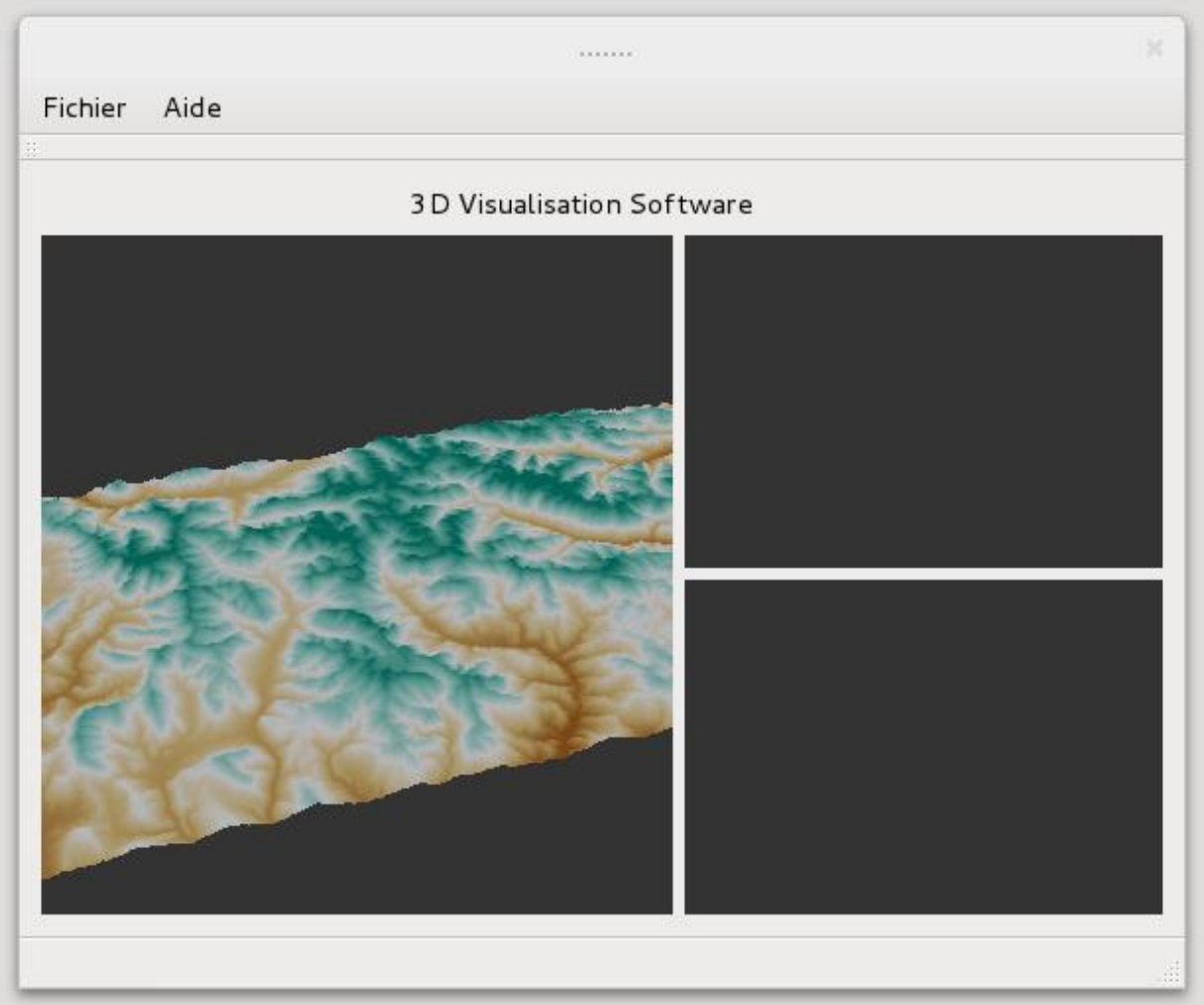

Figure 6 : affichage du MNT avec sa texture

b. Chargement et affichage de l'objet :

Nous avons utilisé les outils disponibles dans la librairie Assimp pour simplifier les opérations de chargement et dessin de notre objet. Cette dernière permet de manipuler plusieurs types de fichiers entre autres **OBJ**, **3DS**, **COLLADA** etc.

Algorithme de chargement :

Le schéma suivant résume l'architecture simplifiée de la librairie :

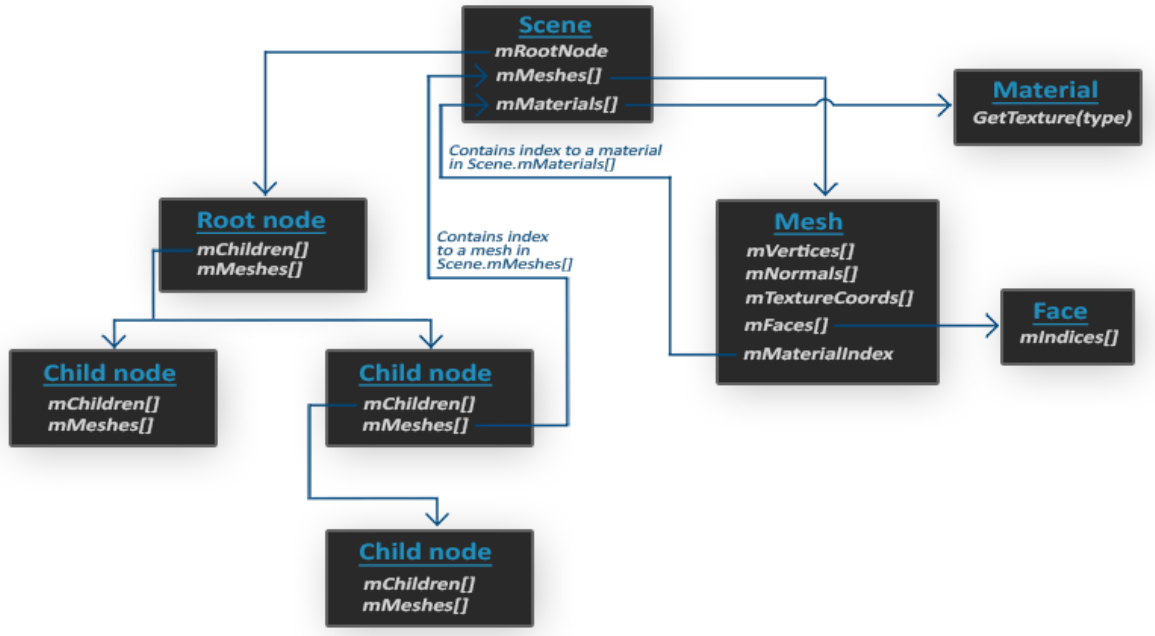

Figure 7: architecture de la librairie Assimp

La méthode **readFile()** de l'objet de la classe **Importer** stocke les données de l'objet 3D dans un objet de type **Scene** .

Nous avons procéder à l'extraction des données qui nous intéressent à savoir les sommets, les normales et les indices contenues initialement dans un des objet de type **Mesh** et **Face** et les ont stocker dans des tableau QVector<float>.

Pour le rendu 3D, le principe est le même que pour le dessin de l'MNT.

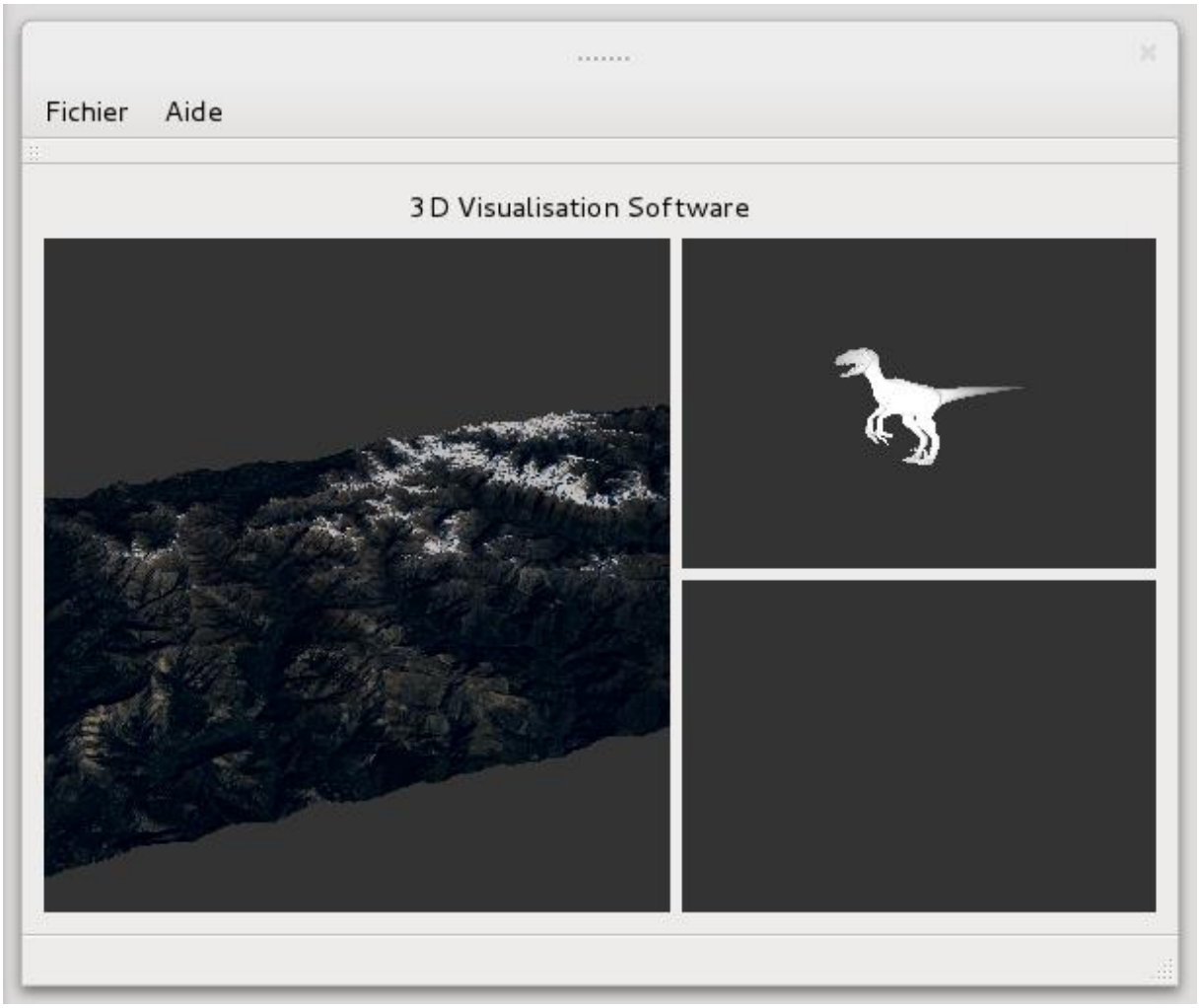

Figure 8 : affichage de l'objet de format Collada.

c. Plaquage de l'objet sur MNT

Algorithme d'intersection de l'objet avec l'MNT:

Notre projet a porté sur des objets rectangulaires et cubiques, l'algorithme d'intersection a pour but de récupérer les points d'intersection des arrêtes de la base avec le relief du MNT, c'est à dire avec les segments des triangles de la maille du MNT, avoir ces points n'est pas une tâche aisée. Il est nécessaire de parcourir en entier notre grille afin que notre algorithme soit capable de plaquer l'objet à n'importe quel endroit du MNT. Une fois les points d'intersection déterminés, on les trie dans l'ordre de position par rapport aux arrêtes de la base avant de les caler à l'altitude du relief, enfin on les stocke dans un tableau d'intersection.

Pour voir plus en détail les opérations réalisées par notre algorithme, nous prenons l'exemple basique d'un segment qui croise le relief, d'abord on vérifie si la ligne est colinéaire avec l'un des segments d'un triangle du MNT, dans ce cas on récupère selon la taille le segment du triangle en question ou bien notre segment. Dans le cas de la non colinéarité on fait appel à une méthode **intersectPoint()** qui permet de récupérer les points d'intersection.

Pour des raisons de simplification, les sommets de l'objet chargé par la librairie Assimp sont récupéré dans un ordre bien déterminer et compatible avec le mode de dessin GL_POLYGONE ou GL_LINES_LOOP. La méthode **plaquerCube()** s'occupe de l'insertion des points d'intersection dans le tableau de sommet destiné à être envoyé au processeur graphique.

Les coordonnées et les dimensions de l'objet sont fixées a priori dans le code et sont ajustés en fonction de l'affichage recherché. Dans un premier temps l'objet est dessiné dans les vrais coordonnés dans la fenêtre **ViewerObjet.**

L'objet plaqué sur MNT est dessiné dans le **ViewerObjetF,** la fonction **SetBoundingBox()** nous permet de définir l'emprise affichable par la camera.

IV. CONCLUSION :

Ce projet nous a été très bénéfique en termes d'apprentissage de code, de gestion de projet et de travail en équipe. Les différents objectifs fixés et recherchés tout au long du temps imparti pour le projet ont été atteints cependant quelques difficultés et perspectives d'amélioration subsistent et sont à signaler:

- Les objets chargés par la bibliothèque Assimp sont rangés dans une structure de tableau très complexe et difficile à cerner ce qui rend leur extraction et reconstruction très difficile.
- L'utilisation de la librairie QGLViewer ne permet pas l'initialisation de la vue en fonction de paramètres variables.
- Le plaquage d'objets complexes constitue un volet d'amélioration.

V. WEBOGRAPHIE ET BIBLIOGRAPHIE :

- Insertion de polyèdres sur un MNT, rapport projet géomatique 2008 , Alexandre BAGUET et Mélanie MORTIER
- http://fearyourself.developpez.com/tutoriel/jeu/Delaunay/
- http://cpp.developpez.com/redaction/data/pages/users/gbdivers/qtopengl/
- http://www.opengl-tutorial.org/
- http://learnopengl.com/

VI. LISTE DES FIGURES :

VII. ANNEXE : DIAGRAMME DE CLASSE