GOULIN
Jolan

CHOUVIN
Sacha

Projet
Synthèse d'images / Programmation et Algorithmique

Rapport

Ce projet fut réalisé en binôme par Jolan GOULIN et Sacha CHOUVIN.

Introduction :

Nous avons dans ce projet réalisé un programme qui, à l'aide d'un QuadTree et différents algorithmes d'optimisation affiche une height map en 3D.

Nous nous sommes répartis le travail de sorte à ce que la mise en commun se fasse de la façon la plus efficace. Néanmoins, en ce qui concerne l'OpenGL, nous avons eu plus de difficultés et avons du nous rendre sur place afin de tester notre code. En effet, l'un étant sur Mac, et l'autre ayant un ordinateur ne supportant l'installation d'une VM, nos essais d'Open GL ont dû s'effectuer sur place, au campus.

Structure du code :

Le code est divisé en 4 dossiers classiques :

- src, contenant les fichiers c++, le fichier .timac et la height map (fichier .pgm)
- include, contenant les fichiers .h
- obj, contenant les fichiers objet
- bin, contenant l'exécutable final

Les noms des fichiers .cpp et .h, décrivent leur fonction. Les fichiers frustum_culling contiennent la LOD.

Partie Programmation Algorithmique :

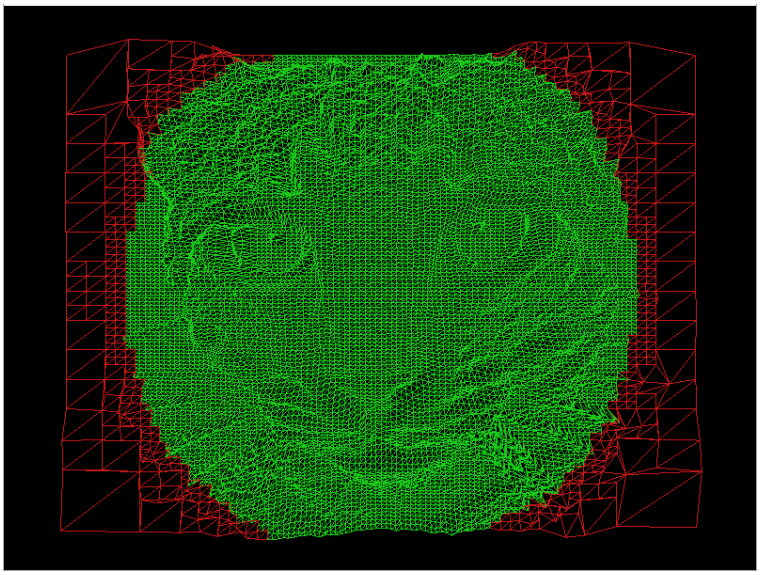
On commence par **lire** mot par mot le fichier test.timac dans lequel sont stockées toutes les données fixes du projet. Ces données sont enregistrées dans la structure tData. On lit ensuite la height map à l'emplacement indiqué dans le fichier .timac afin de créer un tableau des hauteurs des points rangés en fonction de leur indice ([0][0] correspond au point de coordonnées nulles).

Cette height map est stockée dans un **QuadTree** initialisé par une fonction récursive à partir de la height map et de ses dimensions. Chaque node du QuadTree contient sa taille, les coordonnées des 4 points qui déterminent la partie de la height map qu'elle représente, sa profondeur dans le QuadTree si c'est une feuille, et 4 enfants supplémentaires sinon.
Par la suite nous parcourrons le QuadTree de manière récursive et afficheront les informations stockées dans les nodes feuilles sous forme de deux triangles, chacun reliant 3 points de la node.

En ce qui concerne les algorithmes d'optimisation, nous avons un algorithme LOD (Level Of Detail) qui réduit le niveau de détail en fonction de la distance à la caméra de l'objet. Pour chaque node parcourue, il comparera cette distance à la profondeur de la node dans le QuadTree et décidera s'il faut l'afficher ou afficher chacun de ses enfants.

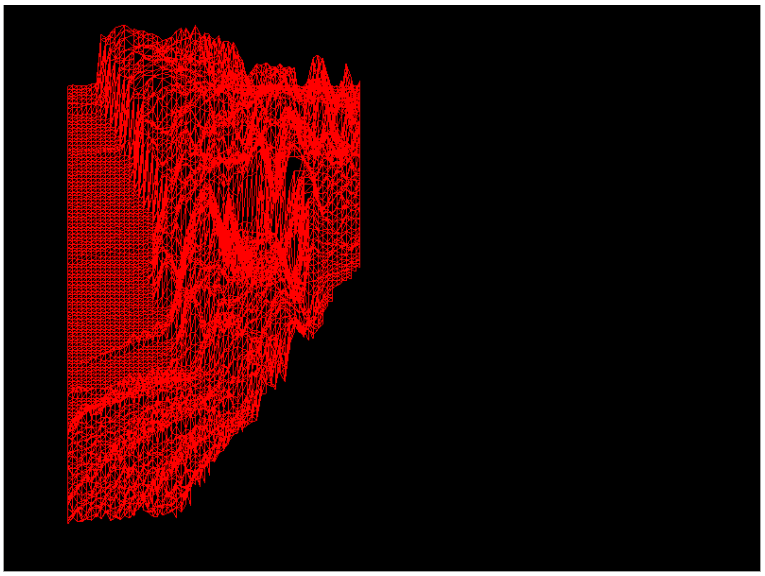
Nous avons jugé préférable de diminuer l'influence du paramètre z dans ce LOD compte tenu de la height map qui présente des changements d'altitudes élevés sur de courtes distances. Nous avons également implémenté un paramètre variable qui gère l'influence de la distance sur le LOD.

Exemple LOD :



Nous avons également un algorithme de frustum culling qui, à l'aide de formules mathématiques, décide d'afficher les nodes présentes dans le champ de vision uniquement et ainsi gagner en efficacité. À cause de multiples changements de systèmes de coordonnées en fin de projet, cet algorithme n'est plus calibré dans la bonne direction.

Exemple frustum culling mal orienté :

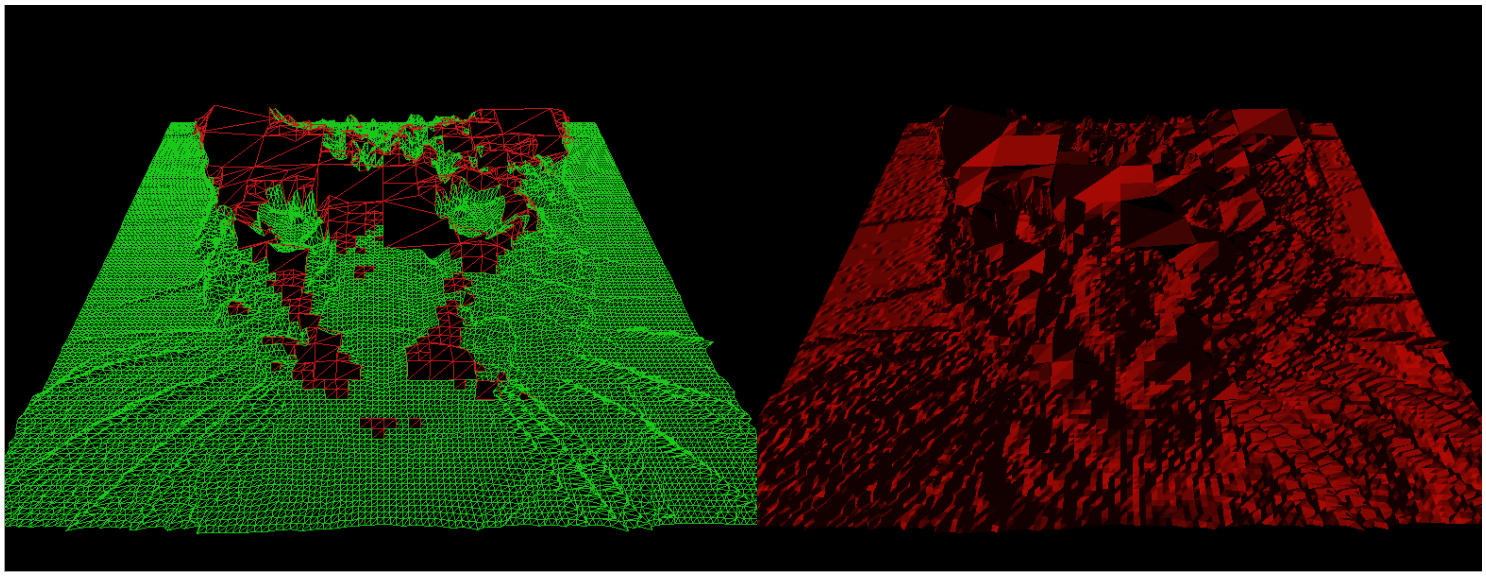


Partie Synthèse d'images :

Nous sommes partis d'un template d'un des TPs pour créer la boucle de rendu et la fenêtre d'affichage.

La fonction drawLinesTriangles permet de tracer les lignes des triangles et drawTriangles de tracer ces mêmes triangles mais pleins. Dans cette dernière, la couleur est modulée par un modèle de Lambert lié à un soleil mobile dans l'espace. Dans le cas de triangles sous formes de lignes, nous affichons les triangles issus de nodes feuilles dans une couleur différente (vert).

La fonction quadtreeTravelDraw, s'appuyant sur les deux algorithmes d'optimisation et de dessin mentionnés plus tôt, s'occupe de tracer à mesure qu'elle parcourt le QuadTree.



Tracé des lignes des triangles

Tracé en triangles pleins

Ces images témoignent de la trop grande influence du paramètre z dans le LOD
(c'est pourquoi nous l'avons atténuée par la suite, cf illustration LOD)

La position de la caméra est définie à partir de ses coordonnées en x, y et z, ainsi par deux angles phi et ro. Nous utilisons ces paramètres pour réaliser deux glRotatef et un glTranslatef. Nous réalisons ensuite un glScalef nous permettant de modifier l'échelle de la map et la rendre plus adaptée au visionnage. Le paramètre z de ce glScalef est modulable via des commandes sur le clavier afin de jouer sur l'intensité des reliefs.

Difficultés rencontrées :

N'ayant réussi à installer des machines virtuelles sur nos PC, nous avons dû nous rendre à l'université pour compiler et tester nos codes. Ne pouvant tester nos codes chez nous, le travail nécessaire pour les débugguer fut d'autant plus conséquent et complexe. Par conséquent, nous avons manqué de temps pour réparer notre frustum culling, ajouter la skybox et la végétation.

Tableau des commandes

i	Déplacement caméra selon +x
k	Déplacement caméra selon -x
l	Déplacement caméra selon +y
j	Déplacement caméra selon -y
p	Déplacement caméra selon +z
m	Déplacement caméra selon -z
z	Rotation caméra vers la gauche
s	Rotation caméra vers la droite
d	Rotation caméra vers le haut
w	Rotation caméra vers le bas
a	Active/Désactive le mode traits
f	Active/Désactive le frustum culling
e	Rotation du soleil (Lambert)
t	Diminue l’intensité des reliefs
g	Augmente l’intensité des reliefs
b	Diminue l'intensité du LOD
n	Augmente l’intensité du LOD

Remerciements à Jules Fouchy, Alaric Rougnon-glasson, Amalia Pluchard.