Modélisation de mondes virtuels
Compte rendu de projet

Alice Malosse - N°141120003285

Janvier 2024

1 Introduction

Ce rapport présente le travail réalisé dans le cadre du porjet du module de modélisation des mondes virtuels. Dans une courte première partie, je discute des problèmes de compilations rencontré avec la base de code fournie avec le TP. La deuxième et troisième partie traite des deux implémentations de génération procédurales de monde : La carte des hauteurs et la génération de route.

Vous trouverez l'ensemble du projet en suivant le lien GitHub ci-dessous. Attention, j'ai du modifier les fichiers de compilation pour pouvoir compiler le projet sans erreur sur mon PC personnel.

Lien vers le code : https://github.com/AliceMalosse/ProjetMMV

2 Compilation

Ce TP utilise la base de code fournis et est compilé sous Windows avec Qt6.
Malgré un suivi rigoureux des instructions de compilation donnés, Window génère une erreur de compilation lors de l'exécution des commandes permettant la copie des shaders dans le dossier de build. N'étant pas parvenue à résoudre cette erreur proprement, j'ai compiler une première fois - malgré l'erreur - pour générer le dossier de build, dans lequel j'ai copié manuellement les shaders. Par conséquent j'ai aussi commenté les lignes de code correspondantes dans le fichier .pro pour éviter d'autre erreur de compilation.

3 Cartes des hauteurs

Pour le développement de la carte des hauteurs, plusieurs classes objets ont été programmé : Box2, Grid, ScalarField et HeightField. L'ensemble des codes se trouvent dans les fichiers height-map.cpp et height-map.h. Certaines des fonctionnalités des objet sont issues des programmes fournis dans le code de départ. Ces programmes ont été adapté au cas précis de l'objet pour lesquelles ils ont été implémenté.
Pour tester les différentes classes des boutons ont été ajouter à l'interface graphique.

Dans un permiers temps, j'ai implémenter l'affichage d'un objet de type Box2. Après résolution de quelques erreurs de programmation, l'affichage fonctionne correctement.

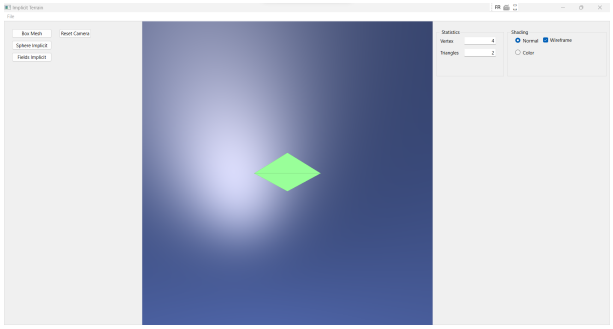

Figure 1 : Affichage d'un carré (objet de type Box2)

Une fois la classe Box2 programmée, j'ai construit la classe Grid. Comme pour la classe Box2, la classe Mesh implémente un constructeur prenant en paramètre un objet Grid. Après implémentation d'un nouveau bouton pour gérer l'affichage de la grille, voici ce que je peux visualiser :

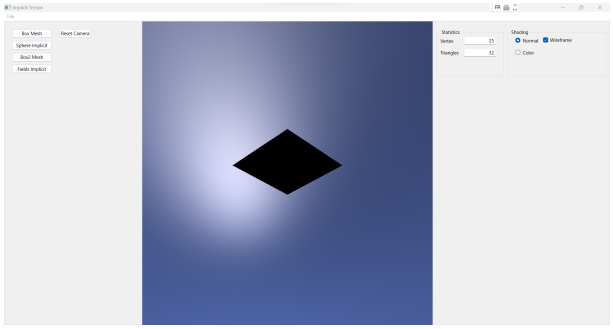

Figure 2 : Affichage d'une grille avec un problème d'affichage de couleur ne permettant pas de distinguer le maillage de la grille.

Le problème de colorisation de la grille n'a pas été résolu.

Ensuite l'implémentation des classes ScalarField et HeigthField suivent le même modèle. Dans un premiers temps ses deux classes sont construite avec hauteurs nulles et peuvent donc être assimilée à des objets Grid.
Pour pouvoir tester les différentes focntionnalité des classes ScalarField et HeightField, je commence par tester les fonctions permettant de charger et de sauvegarder une image de la grille.

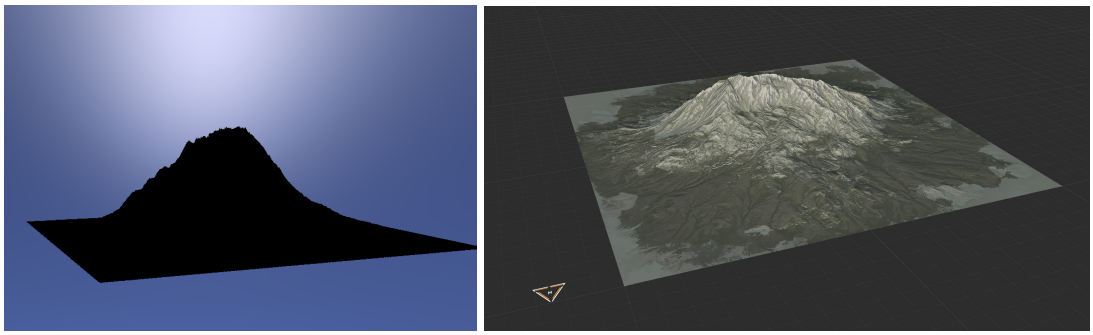

Figure 3 : Rendu non-texturé sur l'application programée à gauche et rendu texturé attendu à droite

Le modèle obtenu correspond à ce qui est attendu. Malheureusement, le problème d'abscence de couleur rends le résultats difficile à visualiser, mais en faisant tourner le modèle on peut comfirmer qu'il n'y a pas d'erreur.
Ensuite, la fonction d'export d'image des hauteurs est programmé, et on peut vérifier son bon fonctionnement en comparant l'image chargée et l'image enresgistrée par le programme.

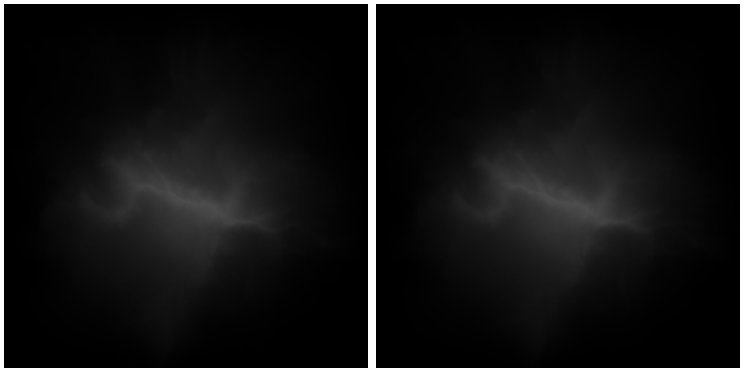

Figure 4 : Image enregistrée (à gauche) et image chargée (à droite) pour générer le scalar field.

On peut constater que les deux images sont en tout point semblables, ce qui nous comfirme le bon fonctionnement de la fonction.

3.1 Problèmes rencontrés

La programmation des différentes fonctions et classes ne m'a pas spécialement posée de problème.
Lors de l'implémentation de mes fonctions dans le fichiers qtemainwindow.cpp afin de pouvoir tester les différentes fonctionnalités, j'ai fait face à un certains nombre de problème de lien externe non résolu. Dans les premiers cas, il s'agissait d'une mauvaise utilisation des variables static. Dans d'autres, j'ai du définir les fonctions dans mon fichier .h au lieu de mon fichier .cpp.
Le problème de coloration à rendu le débuggage des fonctions de scalar field plus complexe. Mais surtout l'implémentation de variation de couleur sur le maillage (application de texture, ombrage) impossible.
Le maximum a été fait malgré ce problème. Je fais de mon mieux pour tenter de corriger dans ce problème, et je l'implémenterais au plus vite dans le GitHub dès qu'il sera corrigé.

4 Génération de route

4.1 Détails d'implémentation

Pour implémenter la génération procédurale de route, je choisie d'implémenter un nouvel objet appelé Road dans les fichiers road.cpp et road.h.
Le détails de l'ensemble des fonctions implémentées pour cet objet peut être consulté dans la partie 5 : Récapitulatifs des fonctions implémentées.

Une route ne pouvant être tracée qu'à partir d'un terrain, les éléments de l'interface utilisateur liés à la création de route ne sont accessible qu'une fois un terrain de type ScalarField chargé.
Les attributs d'une route sont : les point de départ et d'arrivé, exprimés sous forme d'entier représentant l'indice du point dans le grille de terrain, la liste des points formant le chemin le plus optimité et le facteur de calcul des poids. Chaque vertex du terrain posède un certain poids calculé en fonction de sa distance au point d'arrivé et sa pente. Ce poids permet à l'algorithme de Dijkstra de déterminer le plus "court" chemin pour aller du point de départ à celui d'arrivé.
Les fonctions Slope, Gradient et Value sont les même que pour les précédents objets.
Weigth et MinimumWeigth permettent de calculer le poids d'un vertex et de trouver le vertex de poids minimal dans l'ensemble de la grille.
La fonction Dijktra implémente l'algorithme permettant de trouvé le plus court chemin. La boucle s'arrête lorsque le point d'arrivé à été explorer, afin de limiter le temps de calcul de l'algorithme.
La fonction FindPath réccupère les données de l'algorithme de Dijktra pour créer la liste de vertex formant le plus court chemin.

Un fois la route créée et ses attributs définis, j'implémente un nouveau constructeur de Mesh afin de pouvoir lui donner en parametre un objet Road. Dans un premier temps le maillage de la route créée est assez basique. Pour chaque point de la route on crée un second point décalé d'une distance 0.1 selon l'axe x afin de donnée de l'épaisseur à la route.

Pour améliorer le visuel de la route, il faudrait déplacé les points le long de la normale au point. En plus, il faudrait "assouplir" la courbure de la route en calculant une courbe polynomiale passant par les point définis par Dijktra. Mais tout cela n'a pas pu être implémenté.
Voici le type de route que je peux observer avec ma première implémentation :

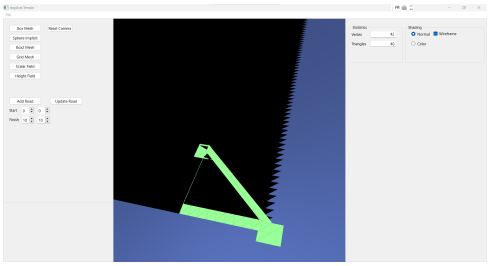

Figure 5 : Première implémentation (incorrecte) d'un route entre les points signalés par des carrés.

Après correction d'une erreur dans la reconstruction du chemin optimal, on obtient un chemin viable. Toutefois, le type de chemin ne varie pas quels que soient les poids appliqué à la pente et à la distane à l'arrivé.

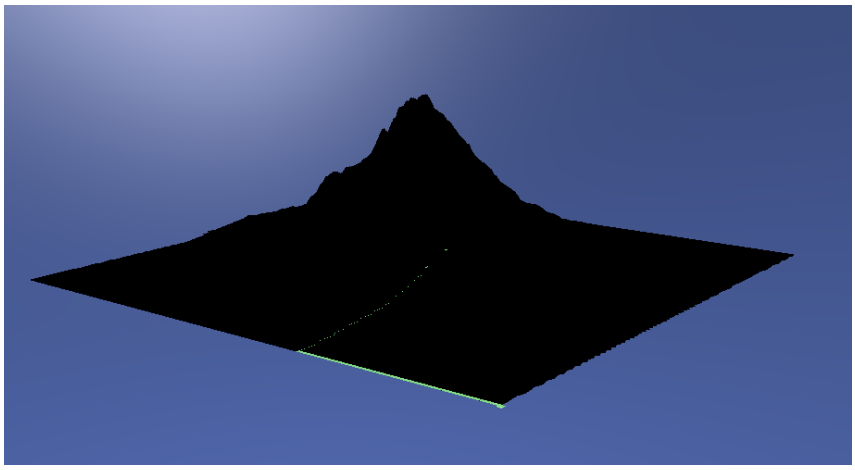

Figure 6 : Chemin pour un facteur 0.5 sur le pente et 0.5 sur la distance

4.2 Résultats

Les résultats visuel ne sont pas très concluant.
L'implémentation de l'algorithme de Dijktra semble correcte mais l'algorithme prend vite beaucoup de temps à tourner. De plus, le plus simple pour vérifier le bon fonctionnement de l'algorithme serait de pouvoir visualiser le résultat produit sur le terrain. Maheureusement, les erreurs d'affichage (ou peut-être viennent-elle directement de l'algorithme justement) ne permettent pas de jugé objectivement de l'efficacité de l'algorithme.
Je continue le debuggage à l'heure de l'écriture de ce rapport, les résultats visible en faisant tourner le code seront donc probablement différents, et je l'espère meilleurs.

4.3 Problèmes rencontrés

Lors de l'implémentation de l'algorithme de Dijktra, j'ai rencontré quelques difficultés à définir le voisinage de mes vertex en me basant sur un voisinnage M_2 comme prévue. Dans un premiers temps j'ai donc limité la complexité de la définition du voisinnage en choisissant un voisinnage 8.

Pour facilité la visibilité de la route générée on ajoute un petit offset sur la hauteur de la route pour que le maillage se détache bien du maillage de terrain.

5 Récapitulatifs de fonctions implémentées

Pour un meilleur aperçus de ce qui a été fait parmis les fonctions demandées dans le sujet, voici un récapitulatif des fonctions et classes implémentées :

Classe Box2 :
- Constructeur : Testé
- Inside : Compile
- Instersect : Compile

Classe Grid :
- Constructeur : Testé
- Index : Compile
- Inside : Compile

Classe ScalarField :
- Constructeur : Testé
- SaveImage : Testé
- LoadImage : Testé
- Gradient : Compile
- GradientNorm : Compile
- Laplacien : Compile
- Smooth et Blur : Non implémenté
- Normalize : Compile
- Clamp : Compile

Classe HeightField :
- Constructeur : Testé
- Height : Compile
- Slope : Compile
- AverageSlope : Compile
- Vertex : Compile
- Normal : Compile
- Shade : Non implémenté
- Export : Non implémenté
- StreamArea, StreamPower, StreamSlope : Non implémenté

Classe Road :
- Constructeur : Testé
- Getter et Setter : Testé
- Slope, Gradient et Value : Testé
- Weigth, MinimumWeigth : Compile
- GetConnection : Testé
- FindPath : Compile
- Dijkstra : Compile