Height Map

Thomas Besson

October 2020

1 Introduction

Le but de ce devoir était de prendre une image de type "heightmap" et de convertir ce fichier en données utilisable par des applications 3D.
Pour lire un fichier "heightmap" j'ai utilisé la librairie "Clmg.h" qui permet de lire les informations du fichier et de retouner les nuances de gris pour chaque pixels.

2 Réalisation

Tout d'abord j'ai créé deux classes pour pouvoir modéliser notre terrain :

1. Une classe "CTerrain"
2. Une classe "CSommetTerrain"

Pour ces deux classes je me suis fortement inspiré du code produit durant la création du cube dans le projet "PetitMoteur3D".

La classe "CTerrain" contient donc plusieurs attributs et plusieurs méthodes. Au niveau des attributs on retrouvera :

1. deux attributs "width" et "height" qui représenterons (sans surprise) la largeur et la hauteur de notre image et qui seront des entiers.
2. un entier "nbrSommets" qui nous dira le nombre de sommet présent dans notre image ou on peut le voir comme aussi le nombre de pixel.
3. un entier "nbrPolygones" qui nous donnera le nombre de polygones que nous allons créer à partir de l'image donné
4. un tableau dynamique "sommets" qui est de type CSommetTerrain à une dimension
5. un tableau de unsigned int "pIndices" qui contiendra les indices de chaque polygone présent dans l'image

Au niveau des méthodes on aura (les méthodes seront détaillées plus tard):

1. le constructeur "CTerrain" qui prendra en argument les deux flottants pour les échelles sur les axes x/y et sur l'axe z, et également le nom du fichier à lire

2. une méthode "CalculerNormale" qui calculera les normales de chaque sommet de l'image

3. une méthode "ConstruireIndex" qui nous construira la liste des polygones et les stockera dans "pIndices"

4. une méthode "Sauver" pour sauvegarder nos données recueilli dans un fichier passé en paramètre

Enfin notre deuxième classe "CSommetTerrain" est identique à la classe "CSommetBloc" développé dans "PetitMoteur3D", on aura donc comme attribut :

1. une position "m_Position" qui est de type XMFLOAT3

2. une une normal "m_Normal" qui est elle aussi de type XMFLOAT3

Je vais donc maintenant expliquer les méthodes de notre classe "CTerrain" qui nous permet de modéliser notre terrain.

2.1 Méthode CTerrain(float dxy, float dz, const char* name)

On ouvre tout d'abord le fichier "name" qui est notre image et on récupère les informations nécessaire pour la création d'un objet 3D, donc la width et la height.
On calcul le nombre de sommets en faisant : width*height
On calcul le nombre de polygone, dont la formule est : (width-1)*(height-1) * 2.
(width-1)*(height-1) nous donnera le nombre de carré dans l'image et on a juste à multiplier ce résultat par 2 car on a 2 polygone dans chaque carré.
Ensuite on initialise notre tableau "sommets" qui est à une dimension et on va y stocker des CSommetTerrain.
Notre tableau est à une dimension donc il faut calculer un indice selon x et y (les coordonnées d'un sommet quelconques) pour pouvoir le stocker dans notre tableau. Pour cela on utilisera la formule suivante : (width * y) + x.
J'ai décidé de parcourir mon image de façon horizontal.
La figure 1 est un image du code pour la création de "sommets".

J'ai mis z en négatif car on est dans un système de "main gauche" donc notre z sera négatif". Notre z correspond donc à la couleur grise que l'on récupère, considéré comme la hauteur. Quand on lis l'image notre point de départ (0,0) est en haut à gauche de l'image et pour nous notre point (0,0) est en bas à gauche.

```
for (int y = 0; y != height; ++y) {
    for (int x = 0; x != width; ++x) {
        sommets[width * y + x] = CSommetTerrain(XMFLOAT3(x * dxy, y * dxy, -img.atXY(x, height - y) * dz));
    }
}
```

Figure 1: Création de sommet

Pour avoir notre image "droite", (le coin en haut à gauche de notre image on le veux en haut à gauche de notre rendu), il faut récupérer l'information, pour le z, au pixel (x, height - y) pour le point se situant en coordonnées (x,y) dans notre "monde".
A la fin de cette boucle on aura donc tous les sommets stocké dans notre tableau "sommets".
De plus les valeurs passé en paramètre, dxy et dz aurons comme valeurs 0.03921f. 0.03921 correspond à 1/255.

2.2 CalculerNormale()

Dans cette méthodes on calcul une normale pour chaque sommet.
Avant toute choses nous savons que le résultat de notre normal sur l'axe z sera négatif car nous somme dans le système "main gauche".
La normal de base sera donc (0,0,-1).

Dans la suite on se basera sur la figure 2 pour avoir une aide visuel.
Imaginons nous sommes arrivé au moment où nous devons calculer la normal de P0 qui se trouve en (x,y). On calcul donc le vecteur

$$\overrightarrow{v_1} = \overrightarrow{P0P1}$$

, le vecteur

$$\overrightarrow{v_2} = \overrightarrow{P0P2}$$

, le vecteur

$$\overrightarrow{v_3} = \overrightarrow{P0P3}$$

, et ainsi de suite jusqu'au vecteur

$$\overrightarrow{v_6} = \overrightarrow{P0P6}$$

On aura donc 6 vecteurs au total pour calculer la normale du point P0.
Après avoir calculer tous les vecteurs nécessaire on calcul le produit vectoriel pour avoir la normal de chaque triangle. Ici on tourne dans le sens horaire donc par exemple la normal du triangle (P0,P1,P2) sera égale au produit vectoriel du vecteur :

$$\overrightarrow{P0P1} \wedge \overrightarrow{P0P2}$$

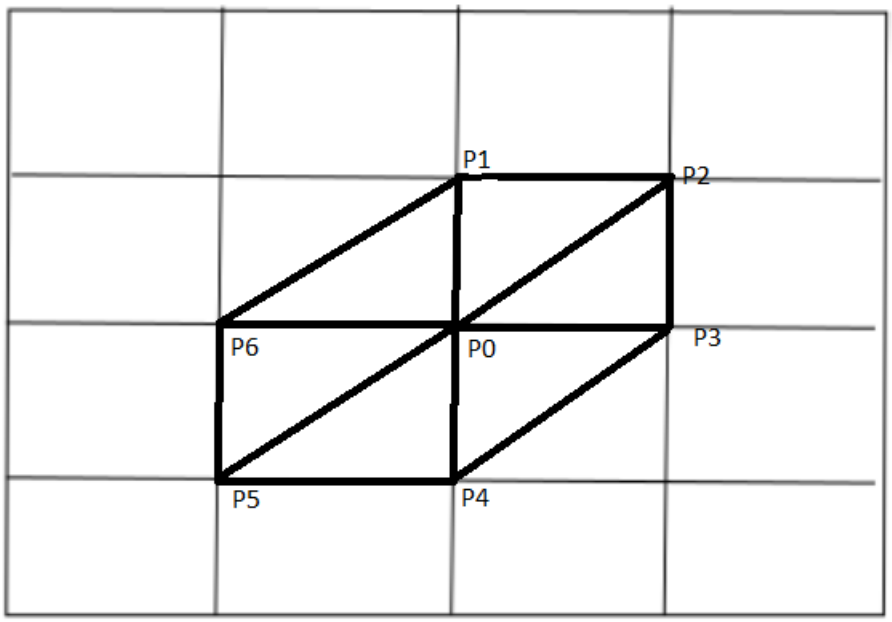

Figure 2: Normal

On additionne ensuite toutes les normales, on normalise la normale résultat et on la stocke dans le sommets P0. Et nous répétons cette opération autant de fois qu'il y a de sommet donc du sommet (0,0) au sommet (width-1, height-1).

2.3 ConstruireIndex()

Dans cette méthodes on va créer les indexes qui nous permettrons de définir nos triangles.

Prenons la situations dans la figure 3. Supposons que on arrive au point I1 qui est aux coordonnées (x, y).
Dans cette image I1 appartient à 2 triangles (si nous ne considérons que les sommets I1, I2, I3, I4).
Nous rappelons que nous tournons dans le sens horaires car nous sommes en "main gauche", donc nos deux triangles crée dans ce cas seront le triangle (I1,I2,I3) et le triangle (I1,I3,I4).
Et pour cela on applique donc les calculs présent dans la figure 4.

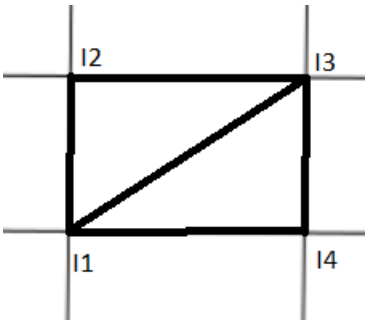

Figure 3: Index

```
int k = 0;
for (int y = 0; y != height - 1; ++y) {
    for (int x = 0; x != width - 1; ++x) {
        pIndices[k++] = y * width + x; //i1
        pIndices[k++] = (y + 1) * width + x; //i2
        pIndices[k++] = (y + 1) * width + (x + 1); //i3
        pIndices[k++] = y * width + x; //i1
        pIndices[k++] = (y + 1) * width + (x + 1); //i3
        pIndices[k++] = y * width + (x + 1); //i4

    }
}
```

Figure 4: Création des Index

2.4 Sauver(const char* file)

Une fois que nous avons créé notre Terrain, calculé les normales et mis en place les index on peut sauvegarder nos données dans le fichier file.
J'ai choisi une structure de sauvegarde similaire à la structure d'un fichier ".obj" car je le trouve claire. On sauvegarde d'abord nos sommets, ensuite nos noramles puis enfin on sauvegarde nos index.
De plus on peux créer un fichier ".obj" et visualiser notre terrain.

3 Amélioration

Bien évidemment nous pourrions améliorer notre processus surtout lors du calcul des normales. Nous pourrions calculer la normales de chaque triangle, les stocker puis aller chercher la normal qui nous intéresse pour un sommet donné. Ceci éviterai de recalculer plusieurs fois les mêmes normales.