Nicolas Luciani
7/03/2023
Master 1 IMAGINE Université de Montpellier

HAI819I : Moteurs de jeux
TP 1 et 2
Compte-rendu

---

1 Introduction

L'objectif de ces TPs est de se familiariser avec la librairie GLFW et GLEW pour modéliser et rendre un terrain.

2 Création d'un plan

Pour créer un terrain, on doit d'abord créer un plan. Pour cela, nous avons implémenté une classe Mesh. Un maillage est ici représenté par une liste indexée de sommets. Une classe Square étends la classe Mesh et permet de définir un plan. L'idée était de paramétrer au maximum la création de notre plan pour assurer la pérennité de notre code. Ainsi, le constructeur de la classe prends en argument :

- 4 points du plan
- la résolution

Pour définir la résolution, il faut d'abord considérer une grille alignée sur les axes, et bornée par les bords du maillage. Le nombre de cellule sur une ligne (ou une colonne) de la grille est la résolution. Une fois le constructeur appelé, il faut mettre à jour les listes de sommets, d'indices des normales ou encore des coordonnées de textures. On prévoit de devoir mettre à jour ces listes en dehors de la classe, nous créons donc une méthode publique adaptée.

```
1  void buildArrays()
2  {
3      /* Deux vecteurs du plan */
4      glm::vec3 bottom = bottomLeft() - upLeft(); glm::vec3 right = upRight() - upLeft();
5      /* Largeur et longueur du terrain */
6      float h = glm::length(bottom); float w = glm::length(right);
7      /* Initialisation des variables de la boucle */
8      /* Vertices */
9      glm::vec3 current = upLeft();
10     glm::vec3 previous = upLeft();
11     glm::vec3 normalizeR = glm::normalize(right);
12     glm::vec3 normalizeB = glm::normalize(bottom);
13     int size = resolution*2; // nb of triangle on one line
14     int nb_v = resolution+1 ;//nb of vertices on one line
15     /* Texture coords */
16     float u,v;
17     u = v = 0.f;
```

```
    float stepJ = w/(float)resolution;
    float stepI = h/(float)resolution;
    float stepU, stepV;
    stepU = stepV = 1/(float)resolution;

    /* Reinitialisation des tableaux */
    m_indexes.clear();m_vertices.clear();m_normals.clear();m_uv.clear();
    m_indexes.resize(6*resolution*resolution);
    m_vertices.resize(nb_v*nb_v);
    m_normals.resize(nb_v*nb_v);
    m_uv.resize(nb_v*nb_v);
    /* Affectation des tableaux en fonction du nombre de sommets */
    for (int i = 0; i < nb_v; ++i)
    {
        v = 0.f;
        for (int j = 0; j < nb_v; ++j)
        {
            m_vertices[i*nb_v+j] = current;
            current += normalizeR*stepJ;
            m_normals[i*nb_v+j] = normal();
            m_uv[i*nb_v+j] = glm::vec2(u,v);
            v += stepV;
        }
        current = previous;
        current += normalizeB*stepI;
        previous = current;
        u+=stepU;
    }
    int index = 0; int nbTriangles = 0;
    for (int i = 0; i < 6*resolution*resolution; i+=6)
    {
        m_indexes[i] = index;
        m_indexes[i+1] = index+1;
        m_indexes[i+2] = index+nb_v;
        m_indexes[i+3] = index+nb_v;
        m_indexes[i+4] = index+1;
        m_indexes[i+5] = index+nb_v+1;
        nbTriangles+=2;
        if(nbTriangles%(resolution*2) == 0) index+=2;
        else index++;
    }
}
```

On peut alors définir un carré avec les positions suivantes
$(-1.0, 0.0, -1.0)$, $(1.0, 0.0, -1.0)$, $(1.0, 0.0, 1.0)$, *et* $(-1.0, 0.0, 1.0)$. On obtient alors le résultat suivant :

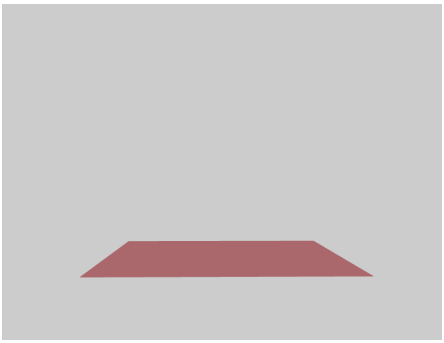


Figure 1: Carré centré sur l'origine du repère monde

3 Textures

Le calcul des coordonnées de textures est donné dans la fonction plus haut, cependant, il faut maintenant charger une texture et s'en servir dans le fragment shader. Il nous est donné une fonction qui permet de charger une texture en format BMP dont nous ne détaillerons pas l'implémentation ici. En revanche, nous ajoutons une variable d'entrée au fragment shader qui lui permettra de gérer les coordonnées envoyées par le vertex shader. On obtient alors le résultat suivant avec la texture donnée "rock".

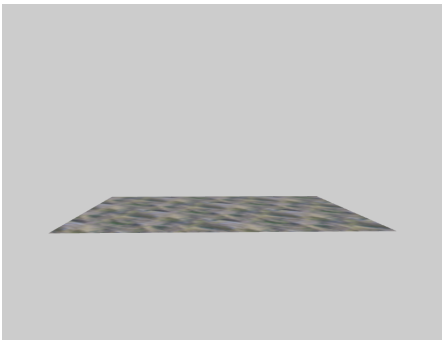


Figure 2: Ajout d'une texture

4 Reliefs aléatoires

Pour créer des reliefs aléatoires, nous avons modifié la fonction présentées plus haut en ajoutant aléatoirement une valeur entre 0 et 1 au la coordonnée y de chaque sommet. On obtient alors le résultat suivant :

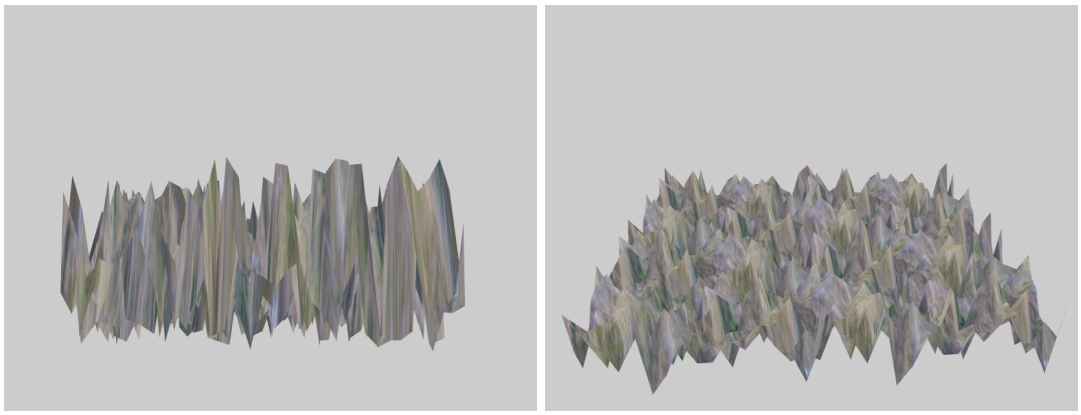


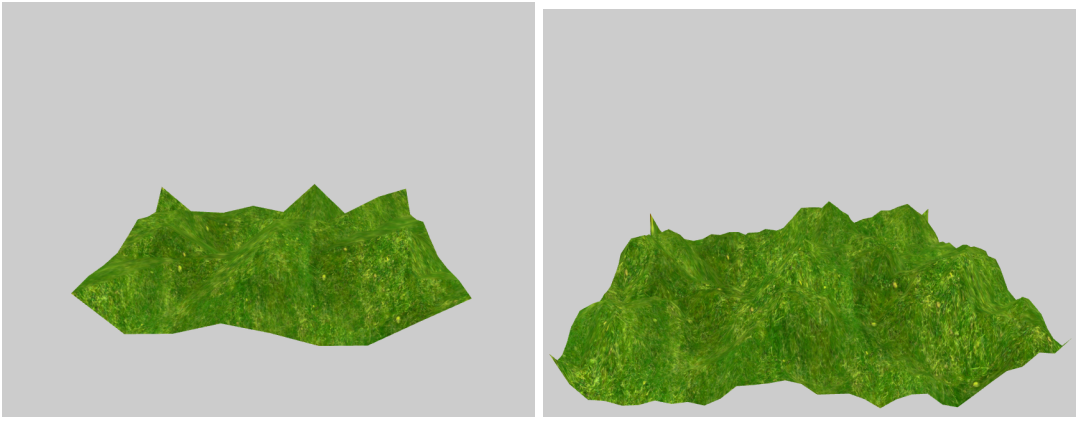
Figure 3: Reliefs aléatoires

5 Carte d'altitudes

Dans l'idée des reliefs aléatoire, nous pouvons maintenant utiliser une carte d'altitude pour modifier la coordonées en y des sommets. Ceci sera réalisé non plus dans la fonction présentée plus haut mais bien dans le vertex shader. En effet, on utilisera les coordonnées de textures envoyées au shader pour modifier l'altitude de notre terrain.

```
#version 330 core

// Input vertex data, different for all executions of this shader.
layout(location = 0) in vec3 vertices_position_modelspace;
layout(location = 1) in vec3 normal_modelspace;
layout(location = 2) in vec2 uv0;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

uniform sampler2D heightmap;

out vec3 o_normalWorld;
out vec2 o_uv0;
out float o_height;

void main(){
        mat4 MVP = projection * view * model;
        mat3 normalMatrix = mat3(transpose(inverse(model)));

        vec4 height = texture(heightmap, uv0);
        vec4 offset = vec4(vertices_position_modelspace.x ,
            vertices_position_modelspace.y + height.r,
            vertices_position_modelspace.z , 1.0);
        o_uv0 = uv0;
        o_height = offset.y;
        o_normalWorld = normalMatrix * normal_modelspace;
    gl_Position = MVP * offset;

}
```

On peut alors obtenir les résultats suivants ; augmenter la résolution du terrain modifie l'apparence du relief.



(a) Terrain de résolution 4 (b) Terrain de résolution 32

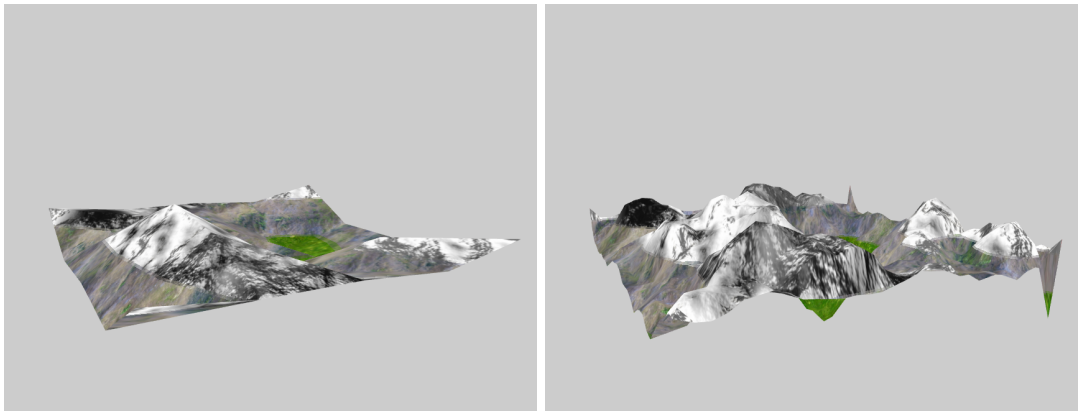
Figure 4: Application de la carte d'altitude Mountain

6 Textures conditionnées

Si l'on veut représenter un terrain, il peut être intéressant que les textures diffèrent d'une zone à l'autre du terrain. Ici, nous avons choisi de changer de texture en fonction de l'altitude. C'est pourquoi nous avons créé une variable de sortie qui permet d'envoyer au fragment shader l'altitude. Nous avons ensuite modifié ce shader pour faire en sorte que la couleur en sortie dépende de cette altitude en entrée.

```
#version 330 core

in vec3 o_normalWorld;
in vec2 o_uv0;
in float o_height;

uniform sampler2D ground;
uniform sampler2D h_ground;
uniform sampler2D heights;
// Ouput data
out vec4 FragColor;

void main(){
        vec4 groundTex = texture(ground, o_uv0);
        vec4 h_groundTex = texture(h_ground, o_uv0);
        vec4 heightsTex = texture(heights, o_uv0);

        if(o_height < 0.5) FragColor = groundTex;
        else if(o_height >= 0.5 && o_height <= 0.51) FragColor = mix(groundTex, h_groundTex, 0.5);
        else if(o_height <=0.7) FragColor = h_groundTex;
        else if(o_height > 0.7 && o_height <= 0.71) FragColor = mix(h_groundTex, heightsTex, 0.5);
        else FragColor =  heightsTex;
}
```

Les fonctions mix utilisées dans certains cas permettent d'adoucir la transition entre chacune des textures, même si le résultats n'est pas encore très satisfaisant.

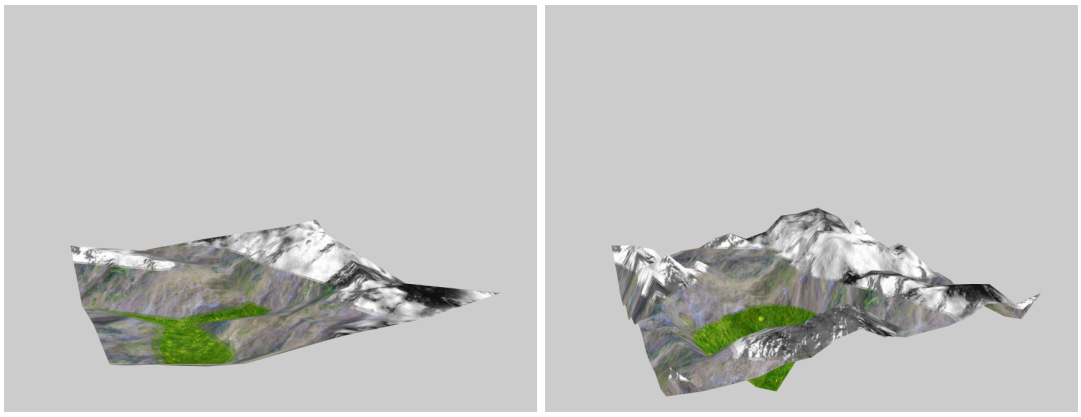


(a) Terrain de résolution 4 (b) Terrain de résolution 32

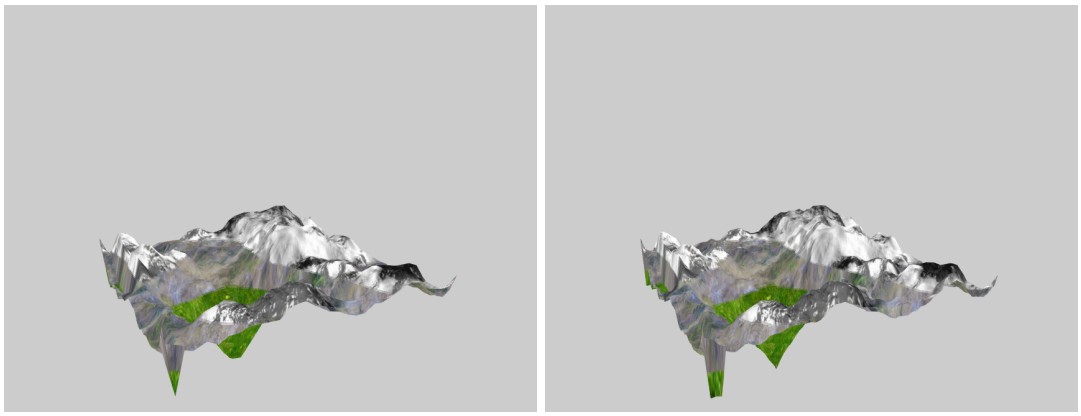
Figure 5: Textures différentes en fonction de l'altitude

7 Résolution du terrain

Comme vu plus tôt, il est possible de modifier la résolution du terrain en appuyant sur '+' ou '-'. La résolution est augmentée ou réduite d'un facteur 2 à chaque modification.



(a) Terrain de résolution 4 (b) Terrain de résolution 16



(c) Terrain de résolution 32 (d) Terrain de résolution 64

Figure 6: Plusieurs résolutions de terrain

8 Caméra

Une première version basique de la caméra était donnée dans la base de code, ou même dans nos TPs précédents. Cependant nous avons ajouté quelques fonctionnalités.

8.1 Déplacement libre

Le déplacement libre de la caméra est le mode initial, il est possible de déplacer la céméra en appuyant sur les touches 'ZQSD' et de la faire pivoter sur les axes de son repère avec un clic gauche et un déplacement de la souris.

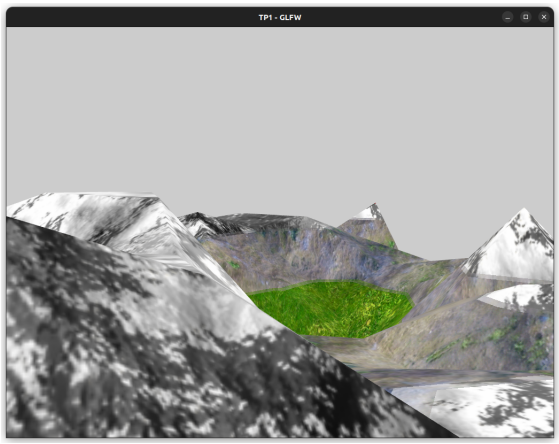


Figure 7: Déplacement libre

8.2 Mode orbital

Il est possible d'appuyer sur 'c' pour activer le mode orbital. La caméra se place alors en $(0, 5, 5$ et regarde le terrain sous un angle de 45 degrés. En appuyant sur la flèche du haut, on augmente la vitesse de rotation du terrain, la flèche du bas la diminue.

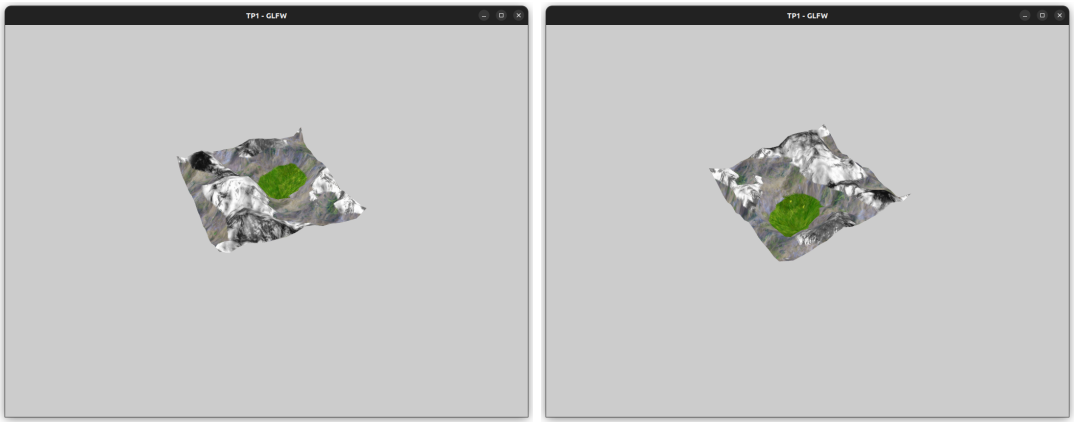


Figure 8: Caméra en mode orbital