Compte Rendu

TP 1&2 Moteur de Jeux

Ayoub GOUSSEM

21 mai 2024

Lien vers le répertoire Git : Cliquez ici

1 Création de la surface

Pour créer un plan, il faut commencer par définir sa résolution et sa taille, ce qui fixe le nombre de points le long de chaque côté et les dimensions globales du plan. Ensuite, on place les points à l'intérieur du plan, espacés régulièrement selon un pas de taille calculé à partir de la résolution.

Connexion des points : Les points sont ensuite reliés pour former des triangles. Pour chaque groupe de quatre points adjacents, on crée deux triangles, ce qui couvre toute la surface du plan.

Préparation des tableaux pour les buffers :

- **Tableau des coordonnées des points :** Contient les positions de chaque point dans l'espace 3D.
- **Tableau d'UV :** Contient les coordonnées de texture pour chaque point.
- **Tableau de normales :** Contient les vecteurs normaux pour chaque point, utilisés notamment pour l'éclairage.
- **Tableau d'indices :** Contient les indices des points qui forment chaque triangle, dans l'ordre nécessaire pour le rendu.

```
void genPlane(std::vector<unsigned short> &indices,
              std::vector<std::vector<unsigned short>> &triangles,
              std::vector<glm::vec3> &indexed_vertices,
              std::vector<glm::vec2> &uv,
              int resolution,
              int size,
              bool randomheight)
{
    indices.clear();
    triangles.clear();
    indexed_vertices.clear();
    uv.clear();

    int nbVertices = resolution * resolution;
    float step = size / (float)resolution;
    float x, y, z;

    for (int i = 0; i <= resolution; i++)
    {
        for (int j = 0; j <= resolution; j++)
        {
            x = j * step;
            if (randomheight)
```

```
            {
                y = std::max((float)rand() / (RAND_MAX), 0.f);
            }
            else
            {
                y = 0;
            }
            z = i * step;
            indexed_vertices.push_back(glm::vec3(x - size / 2.f, y, z - size /
                2.f));
        }
    }

    for (int i = 0; i < resolution; i++)
    {
        for (int j = 0; j < resolution; j++)
        {
            unsigned short bottomLeft = j + i * (resolution + 1);
            unsigned short bottomRight = bottomLeft + 1;
            unsigned short topLeft = bottomLeft + (resolution + 1);
            unsigned short topRight = topLeft + 1;

            triangles.push_back({bottomLeft, bottomRight, topLeft});
            triangles.push_back({bottomRight, topRight, topLeft});

            indices.push_back(bottomLeft);
            indices.push_back(bottomRight);
            indices.push_back(topLeft);
            indices.push_back(bottomRight);
            indices.push_back(topRight);
            indices.push_back(topLeft);
        }
    }
}
```

Affichage des Meshes :

- **Créer les buffers :** Utilisez la fonction `glGenBuffers()` pour créer les buffers nécessaires.
- **Utiliser les shaders :** Activez les shaders avec `gmUseProgram()`.
- **Envoyer les matrices MVP aux shaders :** Utilisez `glGetUniformLocation(programID, "M/V/P")` pour envoyer les matrices modèle, vue et projection aux shaders.
- **Envoyer les différents tableaux aux shaders :**
  - Liez les buffers avec `glBindBuffer()`.
  - Chargez les données dans les buffers avec `glBufferData()`.
  - Activez les attributs des vertex avec `glEnableVertexAttribArray()`.
- **Appliquer les matrices MVP dans le vertex shader :** Dans le vertex shader, appliquez les matrices MVP aux vertices avec `gl_position = P * V * M * vec4(pos, 1)`.

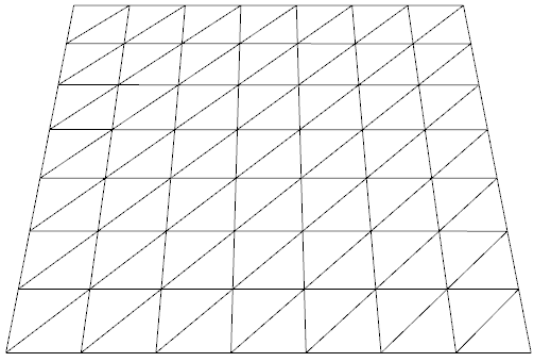

FIGURE 1 – Résolution 8

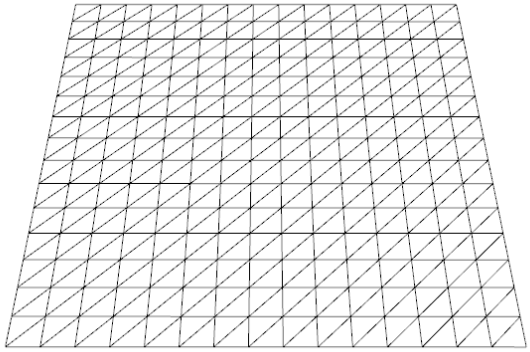

FIGURE 2 – Résolution 16

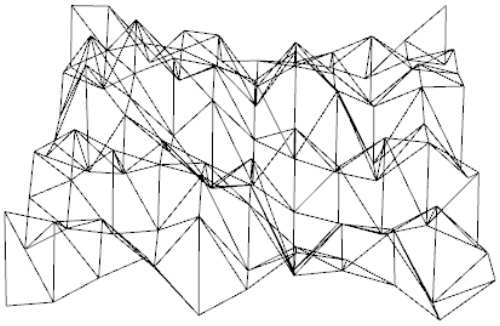

FIGURE 3 – Hauteur aléatoire

2 Gestion de la caméra

La caméra est caractérisée par sa position, son vecteur `UP` et son vecteur `FORWARD`. L'objectif est d'orienter la caméra vers le mesh, ce qui signifie que le vecteur `FORWARD` doit pointer vers le centre du mesh. Pour définir la matrice de vue, on peut utiliser la fonction `glm::lookAt(cam_pos, target, cam_up)`.

La caméra doit orbiter autour de l'objet, ce qui implique que la distance entre l'objet et la caméra reste constante à tout moment. Pour déterminer la position de la caméra en fonction de deux angles, un angle horizontal et un angle vertical, on utilise les formules suivantes :

$$\begin{cases} x = \text{target.x} + \text{radius} \times \cos(\text{vertAng}) \times \cos(\text{horiAng}) \\ y = \text{target.y} + \text{radius} \times \sin(\text{vertAng}) \\ z = \text{target.z} + \text{radius} \times \cos(\text{vertAng}) \times \sin(\text{horiAng}) \end{cases} \tag{1}$$

Pour permettre à la caméra de tourner automatiquement autour de l'objet, il est nécessaire d'incrémenter l'angle horizontal en fonction de `deltaTime`.

3 HeightMap

Pour appliquer la *heightMap*, il est nécessaire d'envoyer la texture au *shader*. Ensuite, dans le *shader*, on modifie la position du sommet en fonction de la couleur du pixel aux coordonnées UV de la texture *heightMap*. La position verticale est ajustée de la manière suivante :

$$\text{pos.y} += \text{texture}(Sampler2D, UV).r \tag{2}$$

Voici une illustration de cette approche :

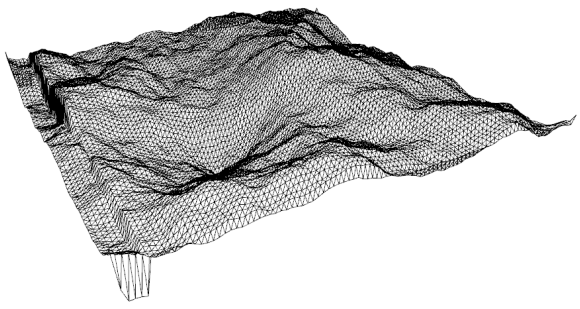

FIGURE 4 – Height Map HM1

4 Gestion de Texture

Pour gérer les textures, il faut :

— Activer un canal de texture en utilisant : `glActiveTexture()`

— Charger la texture dans le canal avec la fonction : `LoadBMP_custom()`

— Envoyer le canal aux shaders avec : `glUniform1i(glGetUniformLocation(programID, "tex"), 0)`. Ce canal sera de type `Sampler2D` dans le shader

— Dans le *fragment shader*, définir la couleur en utilisant : `color = texture(Sampler2D, UV)`

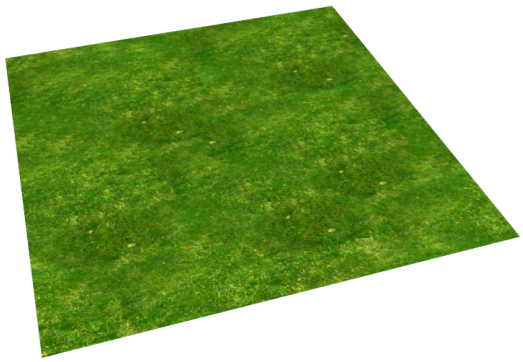

FIGURE 5 – Texture 'GRASS' appliquée au plan

Pour mélanger les textures, nous pouvons utiliser la fonction GLSL `mix()`, qui réalise une interpolation linéaire entre deux textures en fonction d'un poids donné. Les poids peuvent être déterminés à l'aide de la fonction `smoothstep(hauteur1, hauteur2, valeurHeightMap)`, ce qui permet de faire apparaître la texture à partir d'une certaine hauteur.

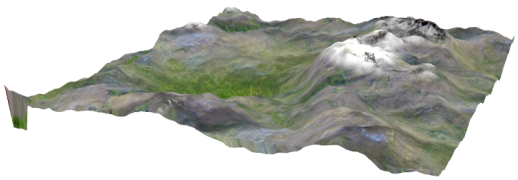

FIGURE 6 – Texture selon l'altitude

5 Commandes clavier

— z,q,s,d,a,e : déplacement libre de la caméra
— 3, 4 : augmenter / diminuer la résolution du plan
— p : activer le mode orbite libre
— o : activer le mode orbite automatique
— r,f : accélérer / ralentir la rotation de l'orbite
— w, x : mode de rendu, polygon / wired