Projet Réalisé par Baurzhan Tair, g1

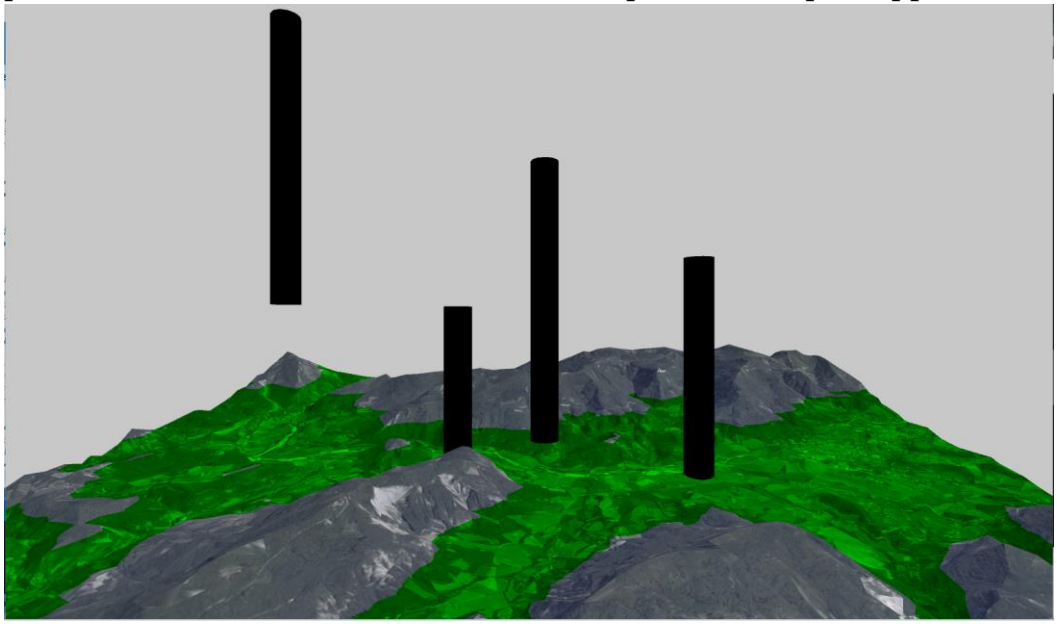
Rapport de projet IGSD 2024-2025

Ce projet a été réalisé par mes soins, en travaillant individuellement. Il consistait à développer un terrain (paysage) en 3D doté de fonctionnalités de navigation et d'effets visuels avancés, tels que la création de pylônes, de lignes électriques et l'utilisation de shaders. Dans l'ensemble du projet, j'ai créé et utilisé 12 fonctions, ainsi que deux fichiers shaders (fragment et vertex), afin d'obtenir des effets graphiques personnalisés sur le terrain. En travaillant seul, j'ai relevé le défi de gérer l'intégralité du processus de développement, ce qui m'a permis d'améliorer mes compétences en programmation.

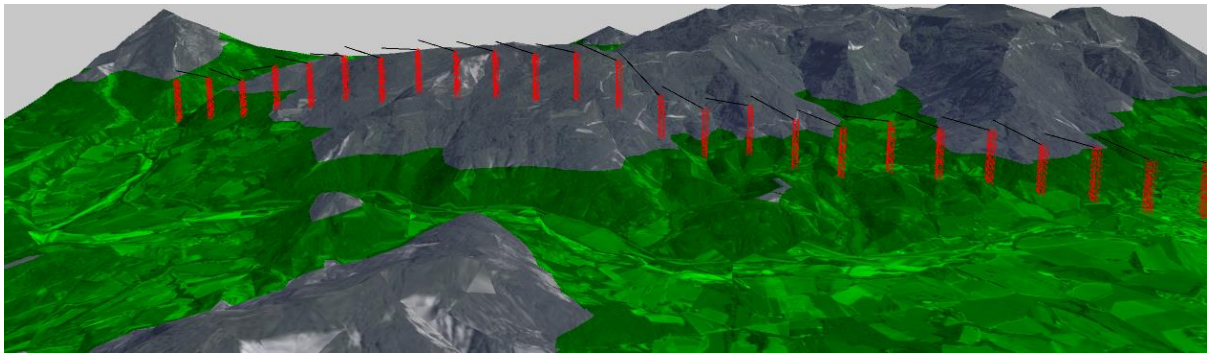
Pour réussir ce projet, j'ai adopté une approche organisée en divisant le travail à faire en deux catégories : les parties faciles, telles que le chargement de données et la création de pylônes, et les parties plus complexes, comme l'intégration des shaders et des lignes électriques qui sont au-dessus de chaque pylône. Travaillant seul, j'ai consacré 2 heures chaque jeudi en travail pratiques, ainsi que 3 heures en autonomie les lundis et les samedis. Ce planning m'a permis de gérer efficacement mon temps et de progresser de manière constante tout au long de ce projet.

Les difficultés rencontrées

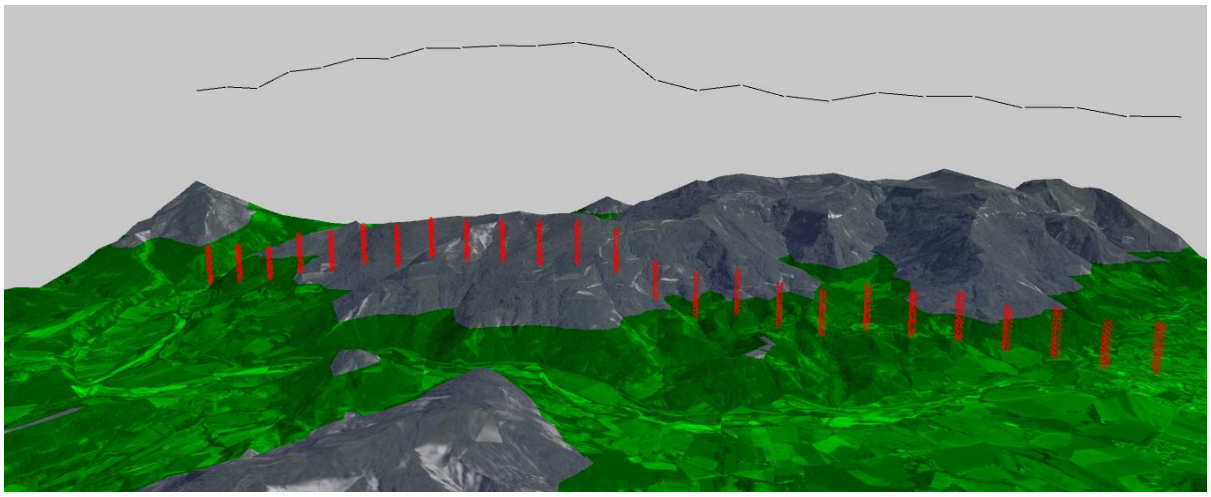
1)La première difficulté est survenue avec la fonction getTerrainHauteur, où j'ai initialement mal calculé les hauteurs, ce qui m'a donné des résultats inattendus. Ensuite, lors de mes essais avec le placement des pylônes, j'ai rencontré des problèmes : ils étaient tous en l'air et mal positionnés par rapport à mes attentes.



2)La deuxième difficulté est survenue lors de l'implémentation des lignes électriques, où j'ai rencontré des difficultés à les connecter correctement entre les différents pylônes.



3)La troisième difficulté était liée à l'emplacement des lignes électriques : elles ne se positionnaient pas correctement au-dessus de chaque pylône en raison d'un problème de calcul de la hauteur maximale pour chaque segment.



Fonctionnalités Implémentées

Dans la partie dédiée aux pylônes, j'ai créé et utilisé 5 fonctions pour gérer différentes tâches dans l'ensemble de projet:

1) La fonction getTerrainHauteur(x, y) parcourt l'ensemble des points du terrain chargé et calcule la hauteur du terrain à la position spécifiée (x et y) en recherchant le point le plus proche dans la géométrie (en termes de coordonnée Z). J'utilise la distance euclidienne pour déterminer le point le plus proche, puis je renvoie la hauteur correspondante Z.

2) La fonction creerPylone() crée la forme d'un pylône en ajoutant des faces au groupe du pylône(en utilisant creerFace avec le param taille d'un pylone).

Chaque face est créée à l'aide de la fonction creerFace(taille), puis cette fonction ajuste la rotation de 90 degrés autour de l'axe Y et la taille du pylône pour obtenir la forme final souhaitée.

3) La fonction creerFace(taille) crée une face du pylône en dessinant des lignes verticales et horizontales à l'aide de la fonction vertex(). J'utilise une boucle pour créer une grille de lignes avec une certaine épaisseur. Les lignes sont tracées à l'aide de la fonction vertex(x, y), définissant ainsi les coordonnées de chaque sommet de cette face.

4) La fonction creerPylone_3(x, y, z) est chargée de créer un pylône à une position spécifiée (x, y, z), en utilisant la forme préalablement définie. Dans cette fonction, je déplace le repère à la position spécifiée, puis j'utilise la fonction shape(pylone) pour dessiner le pylône.

5) La fonction ligneDePylones(nbPylones, distance, r, xDep, yDep) a pour but de tracer une ligne de pylônes entre deux points spécifiés (xDep, yDep) avec un angle donné (r) et une distance spécifique. Pour cela, je calcule les coordonnées de chaque pylône le long de la ligne en tenant compte de la direction et de la distance entre pylones. De plus, je prends en considération les limites du terrain d'après le sujet et je les compare avec les coordonnées finales (xFin et yFin) afin d'éviter que les pylônes ne dépassent les limites du terrain. Enfin, j'utilise la fonction creerPylone_3(x, y, z) pour créer chaque pylône le long de la ligne.

Dans la partie dédiée aux lignes électriques, j'ai créé et utilisé 2 fonctions pour gérer différentes tâches dans l'ensemble de projet:

1) La fonction dessineElectriqueLigne() permet de tracer une ligne électrique entre deux points en ajoutant une variation de hauteur simulant la gravité. Tout d'abord, je calcule la distance horizontale entre ces deux points à l'aide de la fonction dist, puis je définis la gravité de la ligne électrique. À l'aide d'une boucle, je parcours une plage de valeurs allant de 0 à 1 (avec un pas de 0.05) pour interpoler les coordonnées entre les deux points. Pour chaque valeur interpolée, je calcule la nouvelle coordonnée Z en fonction de la distance horizontale et de la gravité, puis j'ajoute un sommet à la forme à cette position.

2) La fonction dessineElectriqueLignes() permet de tracer des lignes électriques entre deux points avec des pylônes intermédiaires. D'abord, je calcule la distance (en utilisant la fonction dist()) et l'angle entre les deux points donnés. Ensuite, en utilisant la fonction ligneDePylones() cf partie (5) , je place des pylônes le long de la ligne entre les deux points. Puis, je boucle sur les pylônes qui sont déjà créés et calcule les coordonnées des extrémités de chaque segment de la ligne électrique. Pour chaque segment, j'obtiens la hauteur du

pylône le plus élevé à chaque extrémité, cad le max. Enfin, je dessine la ligne électrique entre les deux pylônes en utilisant la fonction dessineElectriqueLigne cf partie (1).

Pour la gestion du déplacement, j'ai mis en place 3 fonctions :

1) La fonction mouseDragged(), qui utilise les dernières positions de la souris par rapport aux axes X et Y.

2)La fonction keyPressed(), qui réagit aux appuis sur les touches du clavier(W,S,A,D, 4 flêches, ENTERN et RETURN).

3)La fonction bougeCamera(float x, float y, float z), qui fait évoluer les valeurs des variables de la caméra.

Dans l'ensemble de cette projet, j'ai utilisé 2 fichiers glsl (VertexShader et FragmentShader):

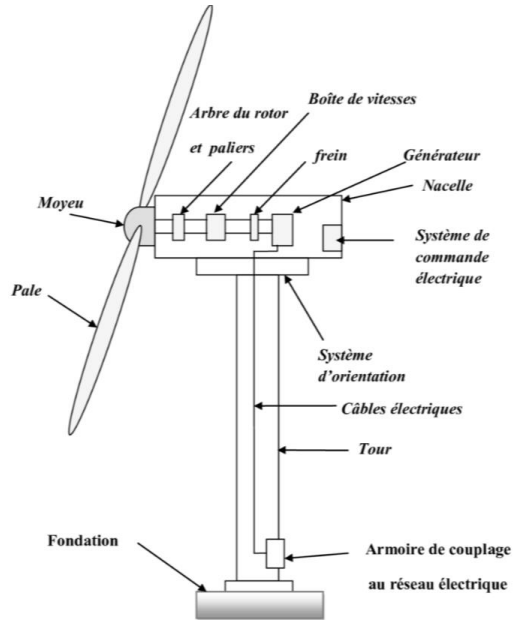
1) Dans le Vertex Shader (VertexShader.glsl), je reçois en entrée les attributs des sommets tels que la couleur, la position et les coordonnées de la texture. J'applique une transformation à la position des sommets en utilisant la matrice de transformation (uniform mat4 transform). Ensuite, je transmets la composante Z de la position en sortie à travers interZ, ainsi que la couleur et les coordonnées de texture interpolées (color et texCoord) pour les transférer au Fragment Shader. Enfin, j'assigne la position transformée à gl_Position, qui est une variable spéciale représentant la position finale du sommet après transformation.

2) Dans le Fragment Shader (FragmentShader.glsl), je reçois en entrée les valeurs interpolées des attributs des sommets, telles que les coordonnées de texture, la couleur et la valeur de Z. Je récupère la couleur de la texture à la position donnée (texCoordOut) en utilisant la fonction texture2D. Ensuite, je calcule le reste de la division de la valeur interpolée de Z par 2 pour obtenir un résultat pour effectuer une vérification en utilisant à la fois la valeur interpolée de Z et le résultat de la division :si Z est dans une plage spécifique et que le résultat est compris entre (0.1 et 0.2), je dessine le fragment avec la couleur noire, sinon, s'il est inférieur à -195, je dessine le fragment avec la couleur verte ou soit je sauvegarde simplement la couleur de la texture par qui est par défaut. Enfin, j'assigne la couleur finale à gl_FragColor avec la couleur finale du fragment qui sera affichée à l'écran.

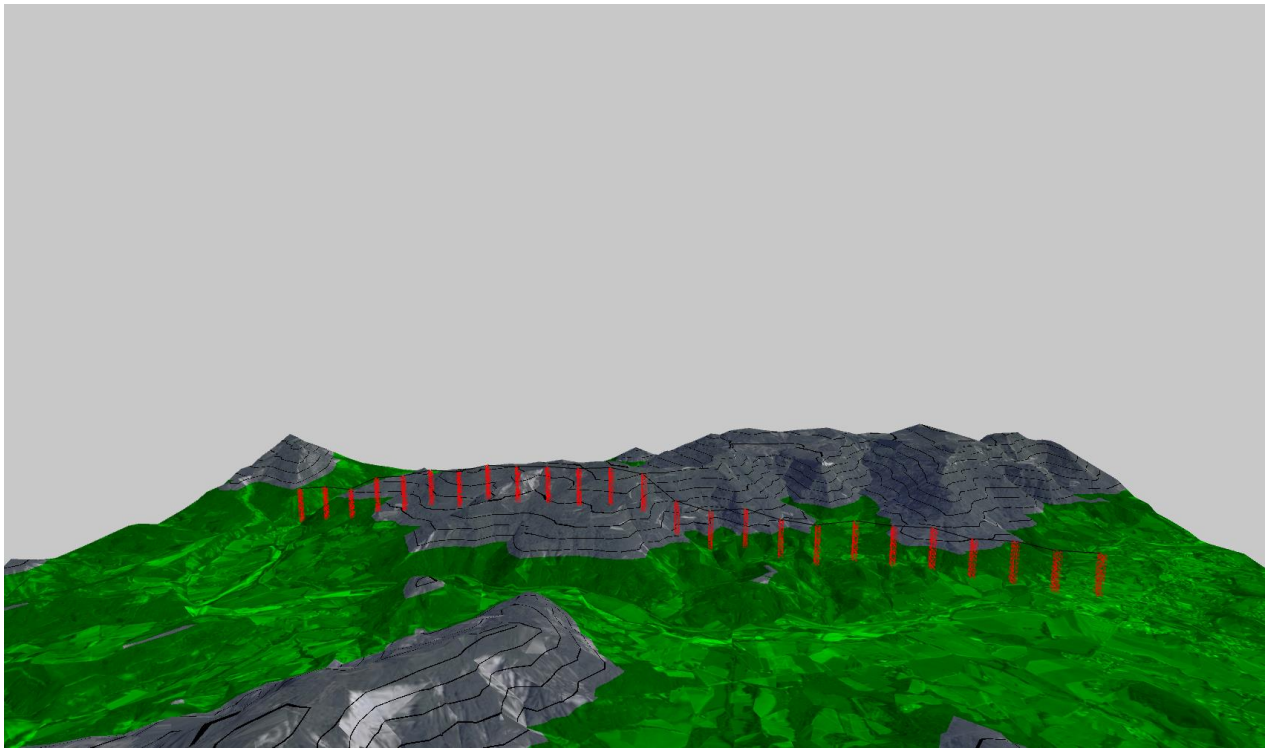
Fonctionnalités Non Implémentées

Dans le cadre de ce projet, je n'ai pas réussi à intégrer les pales d'éolienne au sommet des pylônes. J'ai envisagé d'utiliser la partie principale d'une éolienne

comme le corps d'un pylône(cf la partie dédiée aux pylônes) et d'y ajouter les pales correspondantes. Malheureusement, malgré mes efforts, cette fonctionnalité n'a pas pu être achevée en raison de contraintes de temps et de complexité. Par conséquent, j'ai commenté 3 fonctions concernant aux éoliennes ainsi que toutes les appels de ces 3 fonctions dans mon code.



Le résultat final



À travers ce projet, j'ai approfondi ma compréhension de la programmation des shaders dans le contexte de la visualisation 3D, de la manipulation de textures,

et de la gestion des interactions entre différents éléments de la scène, notamment le chargement du terrain. De plus, j'ai amélioré mes compétences en conception et en résolution de problèmes en créant des fonctions utiles dans l'ensemble du projet, comme le déplacement, la création de pylônes, et de lignes électriques. Pour conclure, ce projet m'a permis d'explorer et d'approfondir différents aspects de la programmation 3D, en particulier la manipulation de shaders, la gestion des textures et des interactions dans un environnement tridimensionnel.

La presentation
https://www.canva.com/design/DAFT5LcXUgo/yuSANdLdDYXtmHqSnPD3cw/edit?utm_content=DAFT5LcXUgo&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton