UE-INF2317M Synthèse D'Image 3D

COTTIER Alexandre – *le code est sur mon lien GitHub :*
https://github.com/Mantador01/synthese-d-image-3D-TP2

1. Concernant la caméra et le déplacement :

De base j'utilisais la caméra avec Orbiter, mais ensuite je suis passée sur une version en mode FPS comme dans un vrai jeu, je mets la position de la caméra au niveau d'une rue a une hauteur de tête à peu près, l'orientation est contrôlée avec les flèches du clavier et le déplacement avec ZQSD / WASD.

A chaque déplacement je ne modifie pas directement la position, je commence par proposer une position candidate (newX, newZ) dans la direction voulue puis j'interroge la grille de navigation (sample_ground(x,z,y,walkable). Ma fonction renvoie l'altitude du sol à cet endroit et un booléen qui dit si la case est bonne ou non.

La grille de navigation est construite à partir du mesh du Rungholt (pour mes test), je parcours tous les triangles, je garde seulement les triangles presque horizontaux et orientés vers le haut et je récupère le matériau associé, je marque la case comme walkable ou non walkable selon le type (pierre, etc.. qui sont autorisé, mais pas l'eau, ou l'herbe par exemple).

Si la case est bonne, j'accepte le déplacement, la caméra se repositionne, si elle n'est pas bonne alors le déplacement est bloqué.

Figure 1 : On voie la vue de la ville depuis le sol, on peut se déplacer tant qu'on reste sur les bloc autorisé (walkable)

2. Rendu direct :

Pour test le pire des cas : beaucoup de lumières ponctuelles dans la scène, je crée 256 lumières aléatoirement dans la ville (position X/Z dans la BB de la scène, hauteur au-dessus du sol et une phase aléatoire pour animer chaque lumière), a chaque frame je fais une petite animation, la lumière flotte un peu en Y et se décale légèrement en X/Z.

Dans le rendu direct, chaque fragment accumule la contribution de toutes les lumières donc ça devient très couteux quand on a beaucoup de fragments visibles et beaucoup de lumières…

Evidemment je ne peux pas montrer l'effet de l'animation des lumières avec les captures d'écran mais si jamais à l'exécution du projet on doit les voir bouger assez joliment.

3. Rendu différé :

Vu qu'on va finir (dans certain cas) avec un problème de performance, on peut mettre en place le rendu différé

Je crée un framebuffer avec plusieurs textures :

- Une texture de profondeur
- Une texture de couleur diffuse
- Une texture normale
- Une texture position

Elles stockent par pixel:

- La couleur diffuse du matériau,
- la normale en espace vue,
- la position en espace vue (ou reconstruite),
- et le Z dans le Z-BUFFER

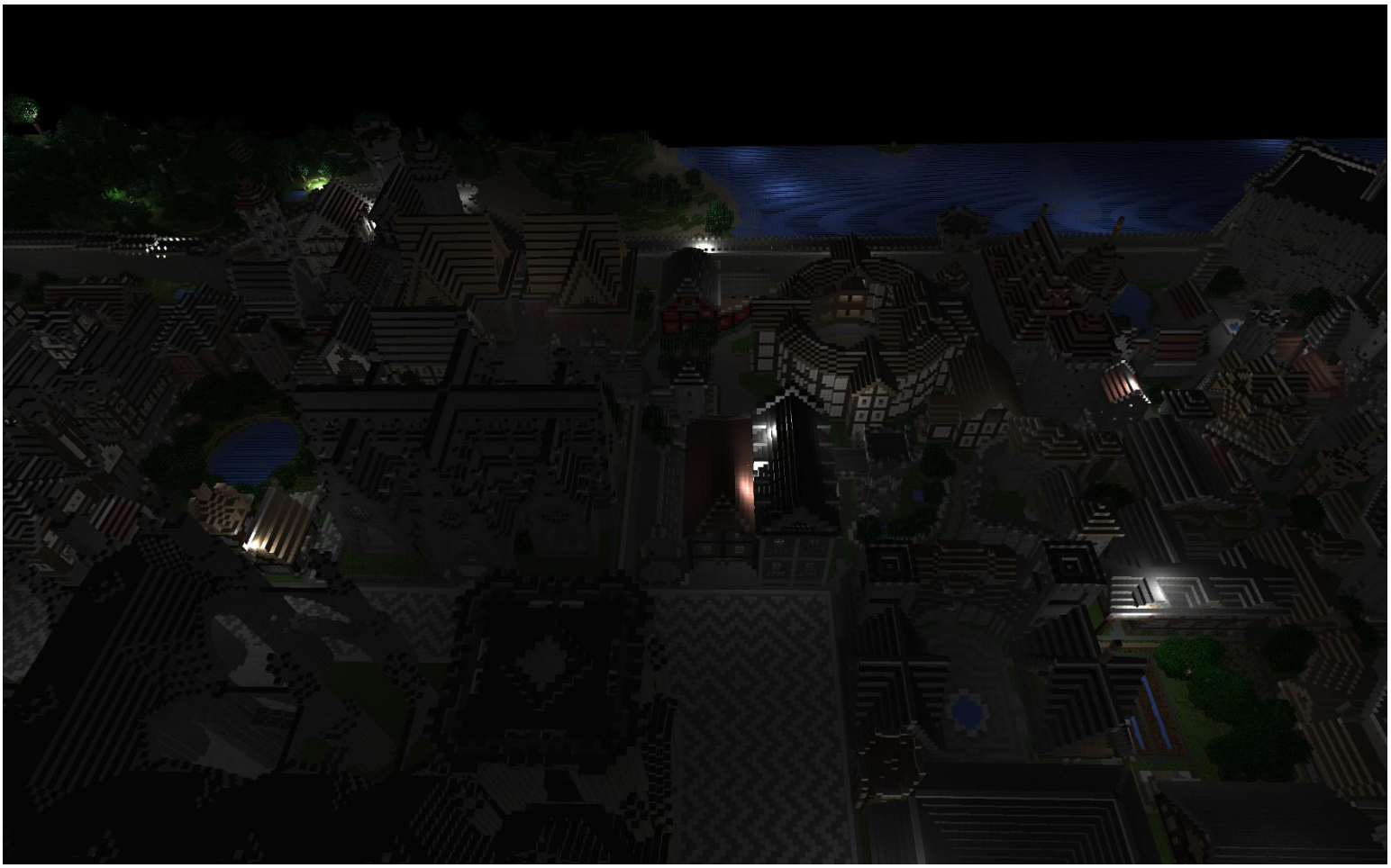
4. Shader G-BUFFER :

Ce shader lit les attributs du mesh (position normal..), applique les matrices et écrits dans les sorties multiples du fragment shader (texture couleur, texture normalisée, position en vue).

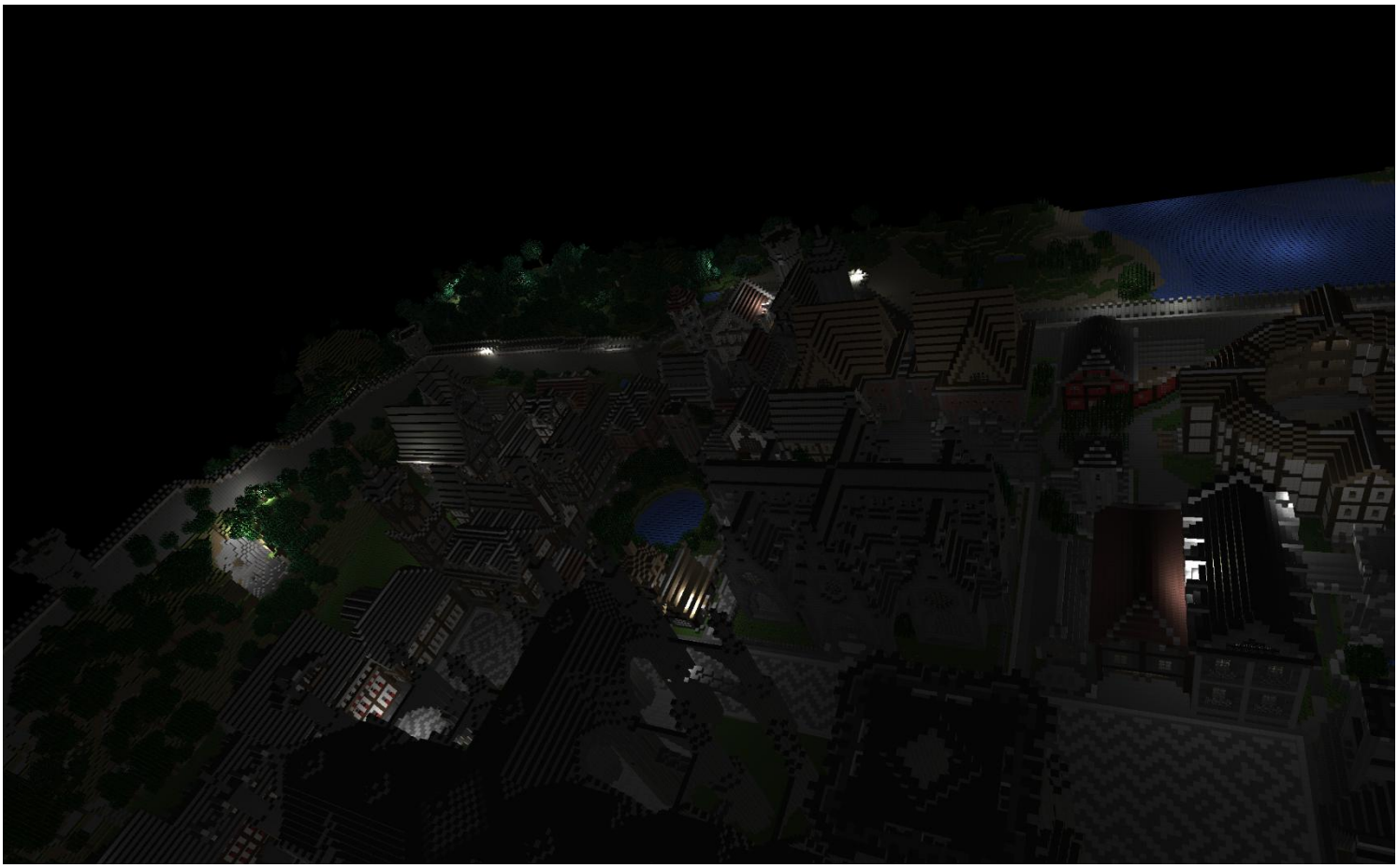
Dans la deuxième passe, je reviens sur le framebuffer par défaut, j'active le shader différé et j'attache les textures du G-BUFFER (uColorTex, uNormalTex, uPosTex), j'envoie les informations sur les lumières (uLightCount, uLightColor) tableau des positions des lumières en espace vue (uLightPos).

Le fragment shader lui lit les informations du G-BUFFER pour chaque pixel, il boucle sur les lumières et calcule la lumière diffuse (Lambert avec max(dot(N,L),0)), accumule le résultat et écrit la couleur finale.

Donc la géométrie est rasterisée une seule fois (pass G-BUFFER), le cout lui dépend de la résolution de l'écran et du nombre de lumières.



A savoir que j'ai pas mal booster certains paramètres surtout pour les tests et le rapport, j'ai fait plusieurs tests j'ai un peu joué avec l'environnement entre autres.



5. Partie optimisation (frustum culling)

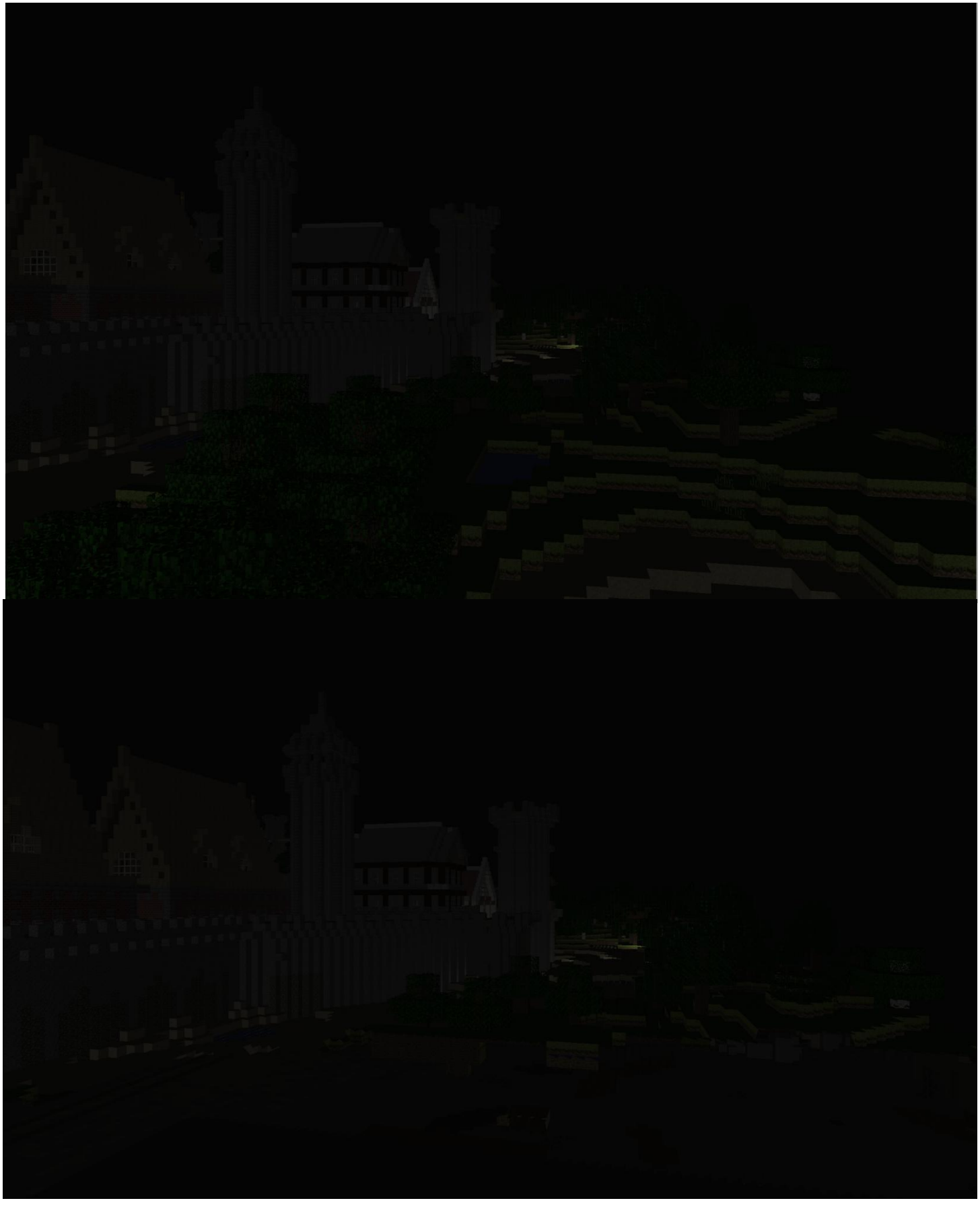
Déjà j'aimerais montrer quelques fail car cette partie ma donner beaucoup de fil a retordre..

Le résultat n'était pas du tout correct (j'ai enlevé du coup), quand je tournais la caméra je voyais que des blocs disparaissait devant moi, c'était vraiment bizarre je ne savais pas d'où venais le probleme.. le principe de disparition de bloc pour pas tout charger marchais bien mais c'est le test de visibilité qui n'étais pas vraiment au point..

Jusqu'à ce que je me rende compte que je calculais mal mon frustum..

Les 2 captures suivantes sont des fails, le bon résultat est juste après !

Je tourne légèrement à gauche et une partie de la map disparait alors que je suis justement entrain de regarder dans sa direction...



Donc en principe déjà un rendu de scène plutôt conséquente comme rungholt est couteux, même avec une bonne carte graphique on aimerait avoir de l'optimisation, le vertex shader doit traiter l'intégralité des sommets a chaque frame même ceux situés derrière la caméra ou très loin.. Donc le but c'étais de mettre en place un frustum culling côté CPU ici (et spoiler alerte côté GPU pour l'option) que les parties de la scène qu'on voie vraiment ou qu'on peut dire pertinente à afficher à l'écran (ça veut tout et rien dire).

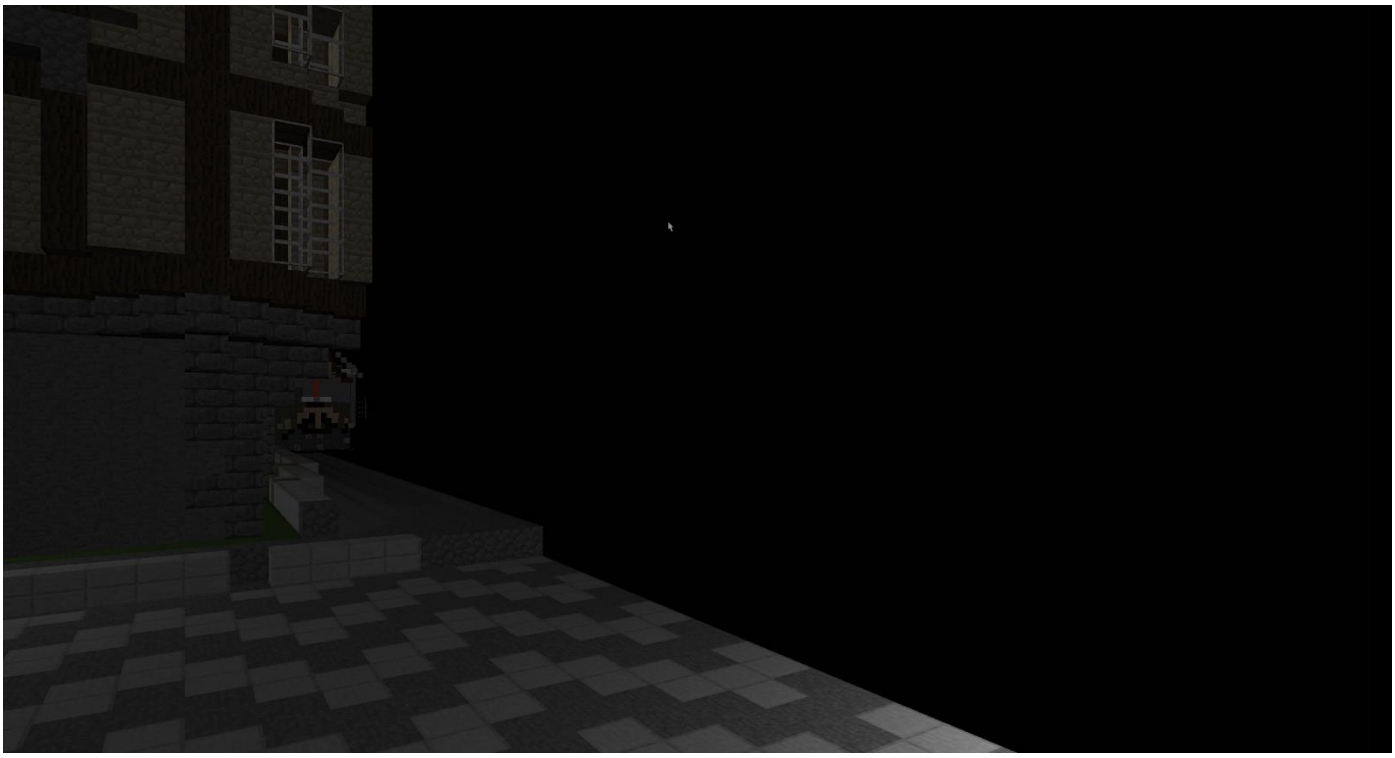
Donc, comme la scène est constitué d'un seul gros mesh, on ne peut pas trier les objets individuellement, on doit donc faire une découpe spatiale "Chunking" a l'init.

Je divise l'espace de la scène en une grille 2D (sur X et Z), chaque triangle de la scène a été assigné à une cellule de cette grille en fonction de la position de son centre, j'utilise la fonction mesh.groups() pour réorganiser les triangles en mémoire pour avoir tous les triangles d'un même bloc contigus dans le VBO (pour le rendu optimisé), puis pour chaque groupe de triangles (chunk) je calcule une AABB définie par ses points min et max.

A chaque frame, dans le render, j'extraie les 6 plans du frustum à partir de la matrice de projection vue MVP, je fais l'algo de Gribb/Hartmann et j'itère sur la liste de tous les blocs (chunks), pour chaque chunk, je teste son AABB contre les 6 plans du frustum, si une boite est entièrement du coté négatif d'un seul plan, alors le bloc est considéré comme invisible.

Si jamais le programme est exécuté, on peut voir dans les logs l'affichage (tous les 2 frames) du nombre de triangles que l'on voit (du moins le frustum), et si on maintient la touche 'c' appuyer on peut voir en déplaçant la caméra (les flèches) que les chunk que l'on ne voyait pas ne sont pas charger (on voit aussi que les logs sont freeze), et si on relâche la touche on les voit réapparaitre.

Là je maintiens la touche ‘c’ puis je tourne la caméra, je vois que le chunk derrière moi n’est pas afficher comme voulu car on ne le voit pas directement.



Ensuite je relâche la touche et je vois tout les chunk se réafficher, évidemment je peux faire des découpes de bloc plus petite, mais voilà.