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

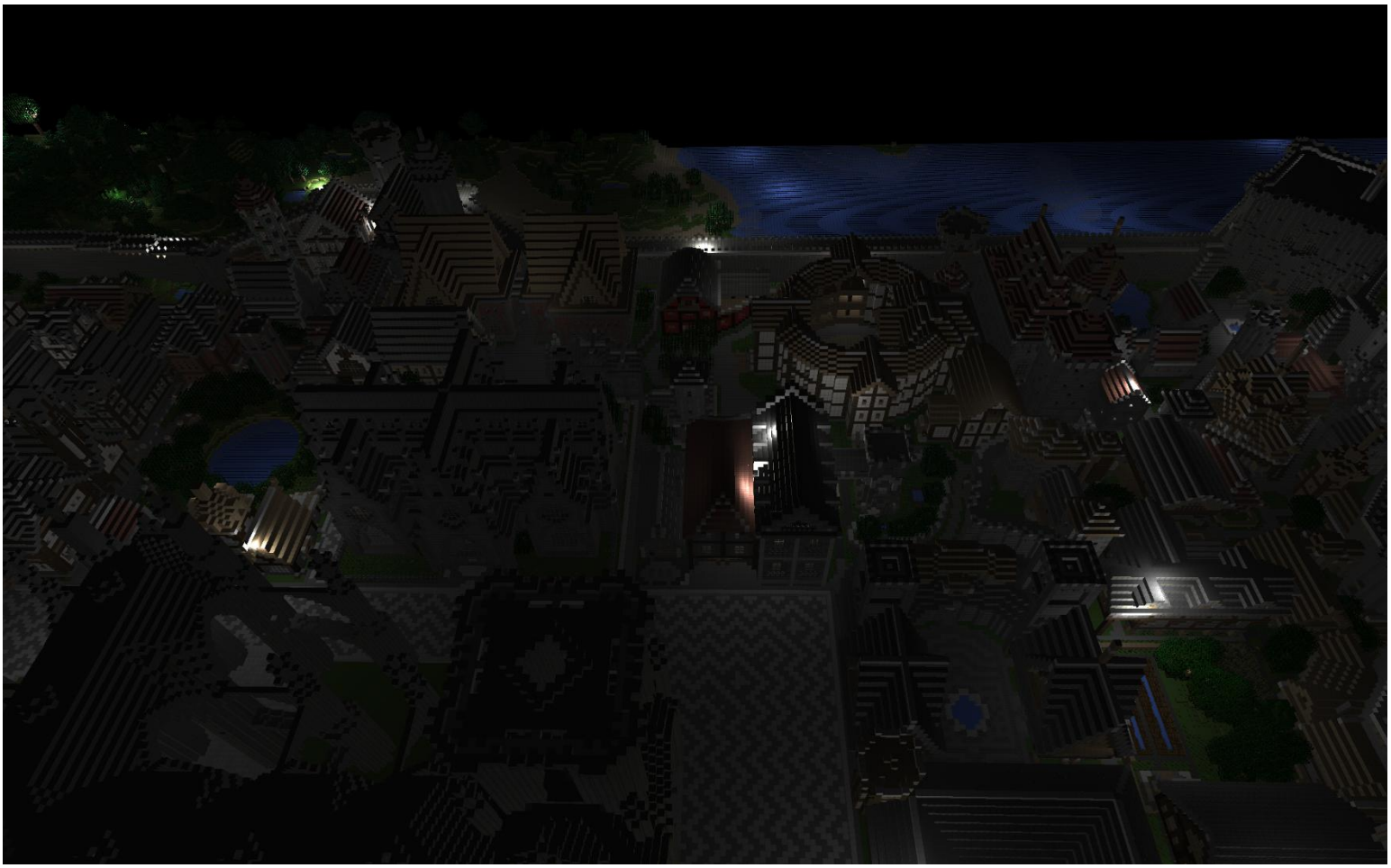
4. Shader G-BUFFER :

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).

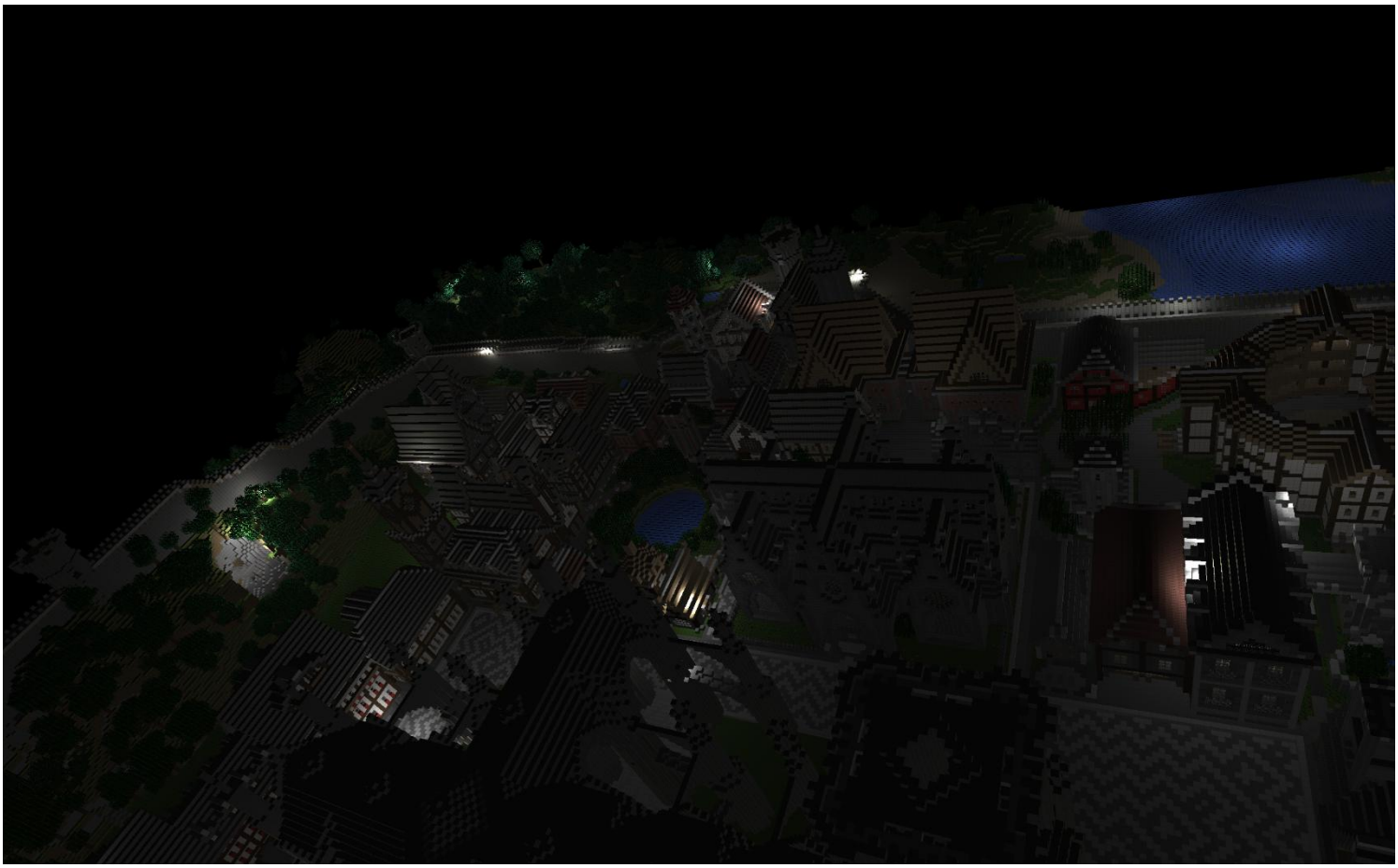
Dans la deuxième passe, je reviens sur le framebuffer par défaut, j'active le shader différé et j'attache les textures du G-BUFFER (uColorTex, uNormalTex, uPosTex), j'envoie les informations sur les lumières (uLightCount, uLightColor) tableau des positions des lumières en espace vue (uLightPos).

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)

J'ai essayé à un moment d'implémenté un système de grille 2D sur le plan XZ qui calcul la boite englobante globale de la scène, ensuite je divise cette boite en une grille 16x16 (enfin j'ai testé avec plusieurs cas), pour chaque triangle du mesh je calcul le centre du triangle, je détermine dans quelle cellule de la grille tombe ce centre et j'ajoute ce triangle dans le block correspondant (un mesh par bloc).

Pour chaque bloc je stocke une boite englobante et le mesh local avec seulement les triangles de la cellule.

Ensuite j'écris un test de visibilité par bloc, je construis le frustum de la caméra à partir de la matrice projection * view, pour chaque bloc je teste sa boite englobante contre ces plans, je génère les 8 sommets de l'AABB, pour chaque plan si tous les sommets sont du mauvais coté du plan alors le bloc est complètement en dehors du frustum donc je ne dessine pas, sinon potentiellement visible alors je dessine.

Sauf qu'en pratique le résultat n'étais 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.. (je suis un peu dégoutée je sens que je n'étais vraiment pas loin).

Je laisse 2 captures d’écran, je tourne légèrement à gauche et une partie de la map disparait alors que je suis justement entrain de regarder dans sa direction…