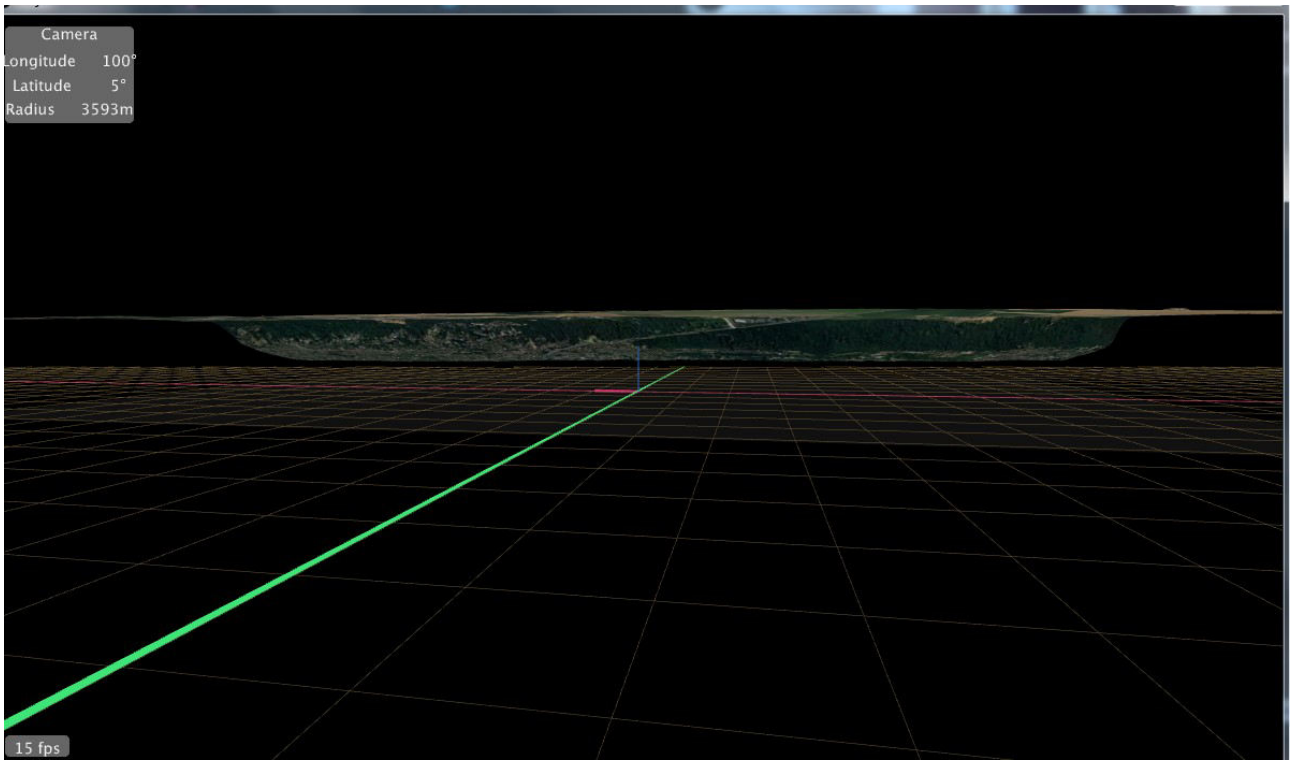
Rapport Projet IGSD de Thomas Combeau

I- Introduction

Le principe de ce projet était de manipuler différentes compétences avec quelque chose de nouveau. En effet, la manipulation de coordonnées à partir de fichiers geojson était quelque chose qu'on n'avait jamais fait. Pour réussir ce projet, il a fallu revoir beaucoup de connaissances accumulées. Dans ce rapport je vais présenter les résultats et les problèmes que j'ai eu par partie. Tout d'abord, j'aimerai préciser que ce projet m'a posé beaucoup de difficulté car j'ai eu des problèmes de performance de mon ordinateur pour afficher notamment les bâtiments. J'ai dû recevoir de l'aide et malgré un temps de lancement long, j'arrive à avoir un rendu. Pendant la présentation orale du projet, il y aura peut-être des problèmes de performance et surtout il y aura un temps de chargement assez important lors de l'exécution du programme.

Partie I : Mise en place de l'espace de travail:

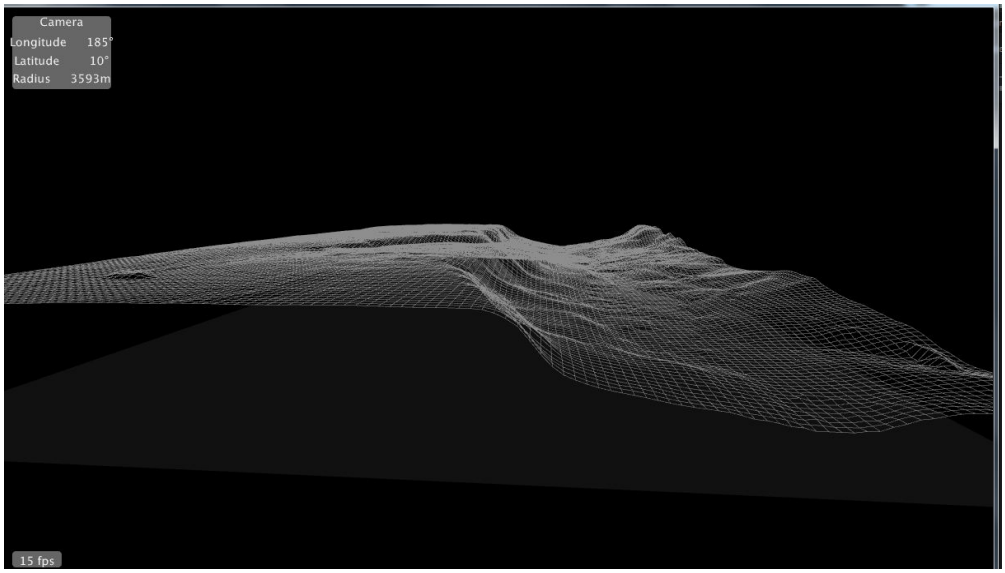
Le but était de réalisé un quadrillage, ou espace de travail, en 3D pour nous permettre de nous orienter facilement pour la suite des travaux. Aussi, on devait réaliser le mouvement de la Camera et aussi l'apparition de deux cases nous signalant la position de la latitude, longitude et du radius dans la première case et l'apparition du FPS dans la deuxième, en temps réel.



En haut à gauche et en bas à gauche on voit les deux cases qui sont implémentées dans la case Hud du programme. On voit bien aussi ici le quadrillage avec la ligne verte indiquant la position de Y, la ligne rouge est quant à elle, la ligne des X et la ligne bleue est le ligne des Z. Grâce à ce nouveau repère, on a plus besoin de travailler avec le repère de base de processing avec notamment Z qui est la profondeur.

Partie 2 : Modélisation du terrain:

La modélisation du terrain fut une prémice de ce qu'allait être ce projet, car sur la moitié du sujet, il a fallu comprendre comment on faisait des conversions de coordonnées : tout d'abord : la conversion Lambert 93 en WGS 84 puis la conversion de conversion Lambert 93 en coordonnées Processing. Ce n'était pas facile au début mais quand il a fallu coder, j'ai pu bien prendre en main ces changements de coordonnées.
Dans un premier temps, il a fallu créer l'ombre portée du terrain avec la Pshape shadow puis il a fallu matérialiser le maillage en fil de fer. A ce moment, j'ai rencontré mon premier blocage du sujet car je ne comprenais pas encore comment fonctionner les changements de coordonnées.



Sur l'image ci-dessus, on voit bien le shadow (l'ombre du terrain) et le wireFrame (maillage en fil de fer). Sur ces 2 Pshape on traverse l'ensembre des coordonnées avec une double boucle for et on utilise nos vertexs. Dans le cas du wireFrame, il faut utiliser ObjectPoint de la classe Map3D pour avoir des coordonnées utilisable sur processing pouvoir réaliser le maillage.

Après ça, j'ai codé la pShape satellite qui permet d'afficher le terrain de Paris-Saclay. Elle part d'une duplication de la Pshape wireFrame mais possède la spécificité d'avoir des coordonnées de texture. Le but est d'utiliser la fonction map() pour avoir les coordonnées où on veut afficher. Cette fonction comme un produit en croix pour passer d'un système à un autre.

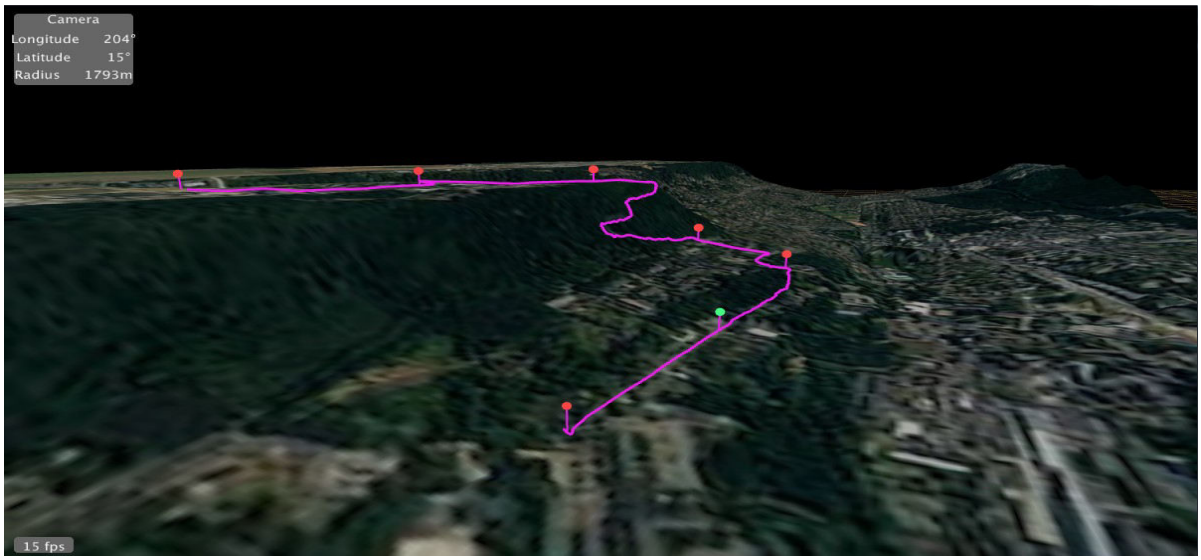
Aussi, à la fin de cette partie il y a l'éclairage à programmer, qu'on peut activer avec la touche L. Et pour avoir une vision du wireFrame, on peut le voir avec la touche W.

Partie III : Affichage d'un tracé GPS:

Cette partie fut la première fois que je manipulais un fichier geojson et l'algorithme permettant de les lire. Au début, ce fut un peu compliqué de bien comprendre le programme mais par la suite j'ai compris qu'au début on test si le fichier geojson est là et est conforme puis qu'on rentre dedans en faisant un switch sur les différentes composantes où on récupère les coordonnées puis qu'on agit en conséquence, par exemple en mettant des vertexs.

Donc dans cette partie, on récupère les coordonnées d'un tracé GPS puis on crée une Pshape track de type LINE_STRIP à laquelle on ajoute des points vertexs qui vont former ensuite une ligne.
On ajoute aussi une Pshape posts et une Pshape thumbtacks pour afficher les têtes d'épingles.

On peut afficher et enlever le tracé GPS avec la touche X.



Sur cette image, on voit une tête d'épingle verte. Par la méthode clic(), si on clique sur une tête d'épingle, elle devient verte. Cette méthode fonctionne par un principe d'exclusion. Aussi, si on clique sur la tête d'épingle tout en bas, il y a une description qui s'affiche disant que c'est l'endroit où se trouve la bâtiment 333. Par contre l'affichage ne dure qu'un bref instant.

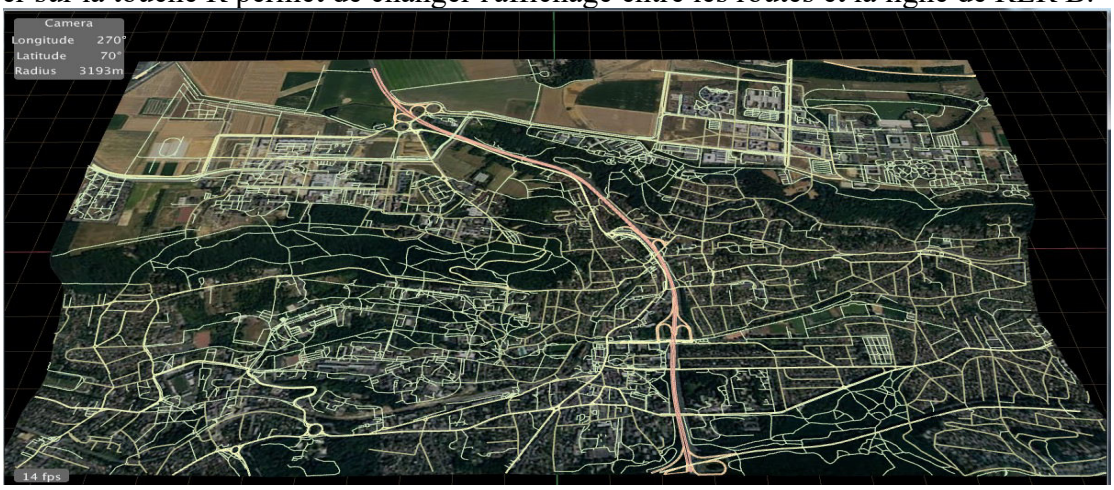
Partie IV : Tracé de voies:

Cette partie fut pour moi très difficile quand il a fallu tracer la ligne du RER B avec QUAD_STRIP. La Pshape lane (de type QUAD_STRIP) se forme à partir de deux cas, le cas où il n'y a que deux tronçons et le reste avec n'importe quel nombre de tronçon. Traiter ces deux cas permet d'éviter des trous dans notre ligne du RER B. Il a fallu aussi comprendre comment prendre 3 points pour appliquer la forme voulue A'A"B'B"C'C".
C'est aussi la première fois qu'on se sert d'une shape de type GROUP, cela permet d'ajouter plusieurs Pshape à ce groupe et permet un affichage d'un ensemble de Pshape assez facilement, car dans la méthode on a plus qu'à mettre à jour l'état de ce groupe de Pshape et pas chaque Pshape un par un.
Il faut aussi précisé qu'à chaque vertex, ce vertex est bien dans notre terrain, on utilise alors la méthode booléenne geo.inside() ou obj.inside() (en considérant geo comme un objet GeoPoint de la classe Map3D et obj un objet ObjectPoint de la même classe). On met aussi une élévation pour éviter certains problèmes et que les affichages ne soit pas dans le sol de notre terrain.

Sur cette image, on voit bien comment est faite la construction de la ligne RER B.

Ensuite, dans une classe Roads, on va chercher à généraliser cette méthode. La différence est qu'on a différents types de route et de caratéristiques propre à chaque type de route, elles sont définies par leur largeur, leur élévation et leur couleur. J'ai alors généralisé dans une méthode traceLane(), la façon de faire une route selon ses caractéristiques mis en paramètre de la méthode.

Appuyer sur la touche R permet de changer l'affichage entre les routes et la ligne de RER B.

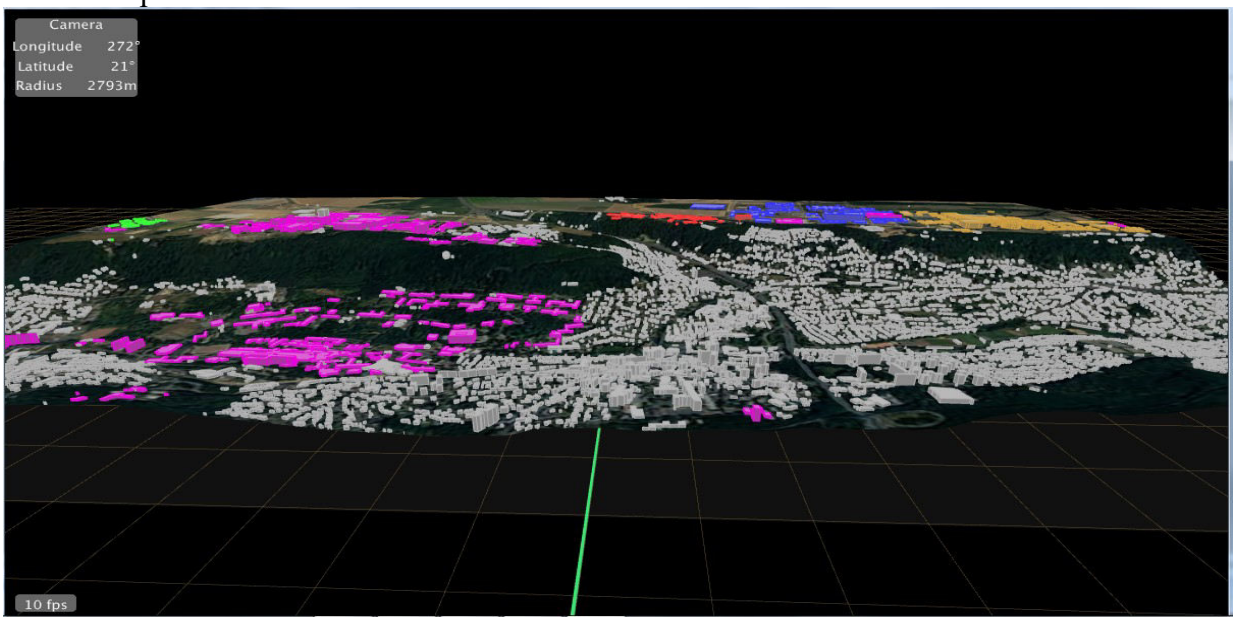


Sur cette image, toutes les routes sont tracées.

Partie V: Tracé de bâtiments:

Cette partie possède beaucoup de similarités avec les précédentes sur la façon dont elle récupère les coordonnées et la façon dont elle affiche les Pshape. Mais ici, on traite de nouveaux type de Pshape, Quad pour la Pshape walls et Polygon pour la Pshape roofs. C'est aussi différent car cette fois on ne construit pas les Pshapes dans le constructeur, mais dans la méthode add(). Ceci permet de récupérer plusieurs fichiers geojson de différents types de bâtiments et de les afficher tous en même temps tout simplement en faisant plusieurs appels à la méthode add() dans le setup() de la classe principale (ici la classe Projet donc).

La touche B permet d'afficher les bâtiments.

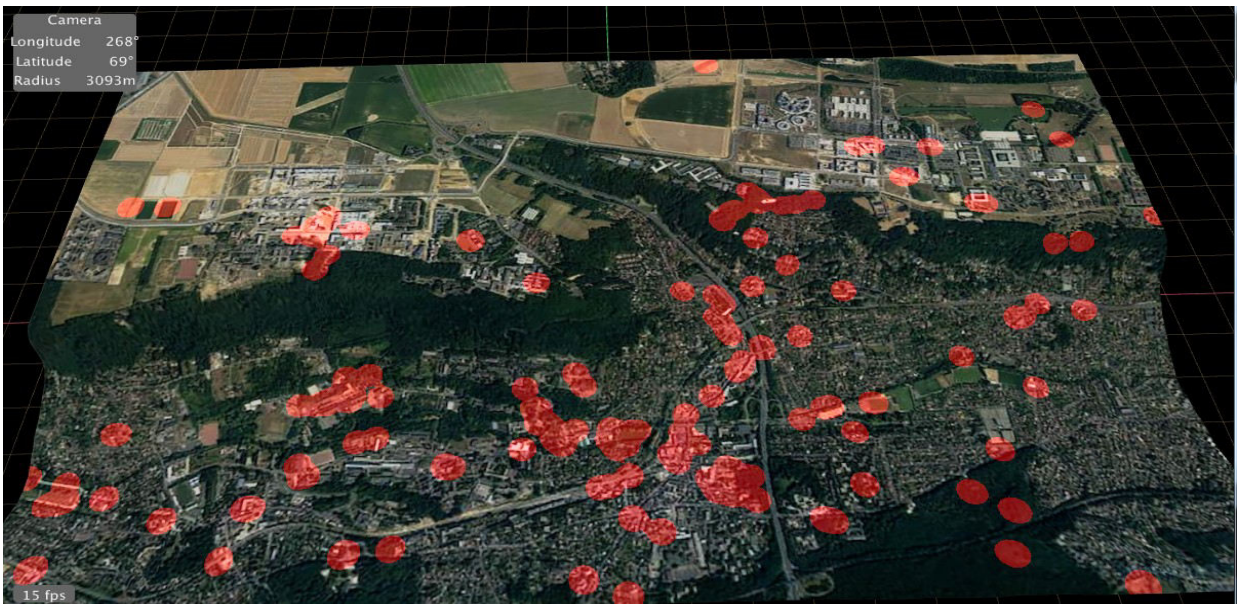


Les bâtiments en gris m'ont causés des problèmes de performance pendant le chargement car le fichier geojson correspondant est lourd.

Partie VI: Réalisation de « cartes de chaleurs »:

Cette partie concerne l'utilisation de shaders, pour mettre de la couleur dans les zones données. Dans mon projet, je n'ai pris que les coordonnées concerant les zones de picnic.
Dans cette méthode, le principale était d'abord de créer une méthode getPoint() qui permet d'avoir une liste de Pvector qui correspondent aux points d'intérets, c'est à dire les points récupérés des coordonnées du fichier geojson des zones de picnic, cette méthode se trouve dans la classe Poi.
Ensuite, il a fallu calculer les distances entre les points d'intérets de cette liste et les points des sommets des vertex du terrain, c'est-à-dire la Pshape satellite de la classe Land; puis il faut attribué la plus petite distance pour chaque point de satellite vers les shaders avec la méthode setAttrib(). Dans mon projet, j'ai attribué les distances vers les shaders sous le nom de heat.
La méthode distPointInteret() se trouve dans la classe Land et est appelé dans le setup() de la classe principale.

Ensuite, dans les shaders, j'ai modifié la couleur selon la position du point x de vertHeat (inférieur à 60 par rapport à la distance) en ajoutant 0,5 à gl_FragColor.r dans le ficher FragmentShader.glsl.
On peut noter aussi que l'appel au Pshader dans le draw est avant l'appel à la méthode update() de Land et se referme après avec resetShader(), c'est pour éviter que le shader s'applique sur le reste du programme.



Si on veut afficher sans cette carte de chaleur, il faut mettre en commentaire les lignes « shader(myShader); » et « resetShader(); » dans le draw() de la classe principale.
Aussi, les cartes de chaleurs pausent un problème au niveau de la lumière qu'on peut activer avec la touche L. Il faut donc mettre en commentaire les lignes dites précédemment si on veut voir la carte avec la lumière.

Conclusion:

Ce projet fut pour moi très long à réaliser et je fus resté bloquer sur différents problèmes souvent plusieurs heures. Mais ce projet m'a permis de mettre en action les connaissances accumulés cette année en IGSD et fût une bonne expérience pour appliquer un spectre large de compétences.