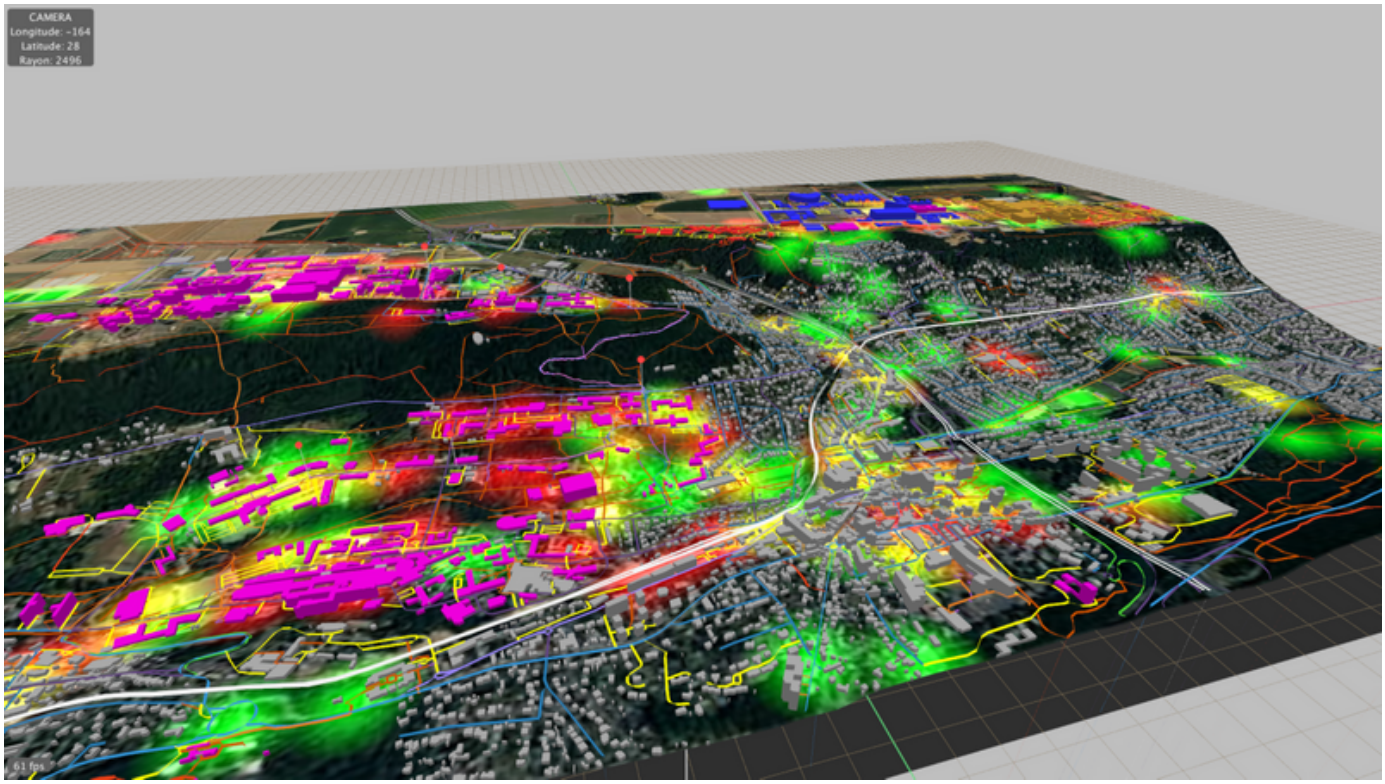
Anaëlle COSSARD
Alice PETIOT

Rapport Final du Projet d'Informatique Graphique pour la Science des Données

Objectif :

Réaliser une représentation en 3D du site de l'Université Paris-Saclay.



Université Paris-Saclay
2020-2021

Table des matières

I. Introduction

Cette année dans le cadre du cours d'Informatique Graphique pour la Science des données, nous avons dû réaliser une représentation en 3D de l'Université Paris-Saclay. Pour ce fait, nous avons tout d'abord mis en place l'espace de travail à l'aide d'un gizmo comme repère qui a permis de s'orienter et d'implémenter correctement la caméra. Dans un second temps, nous avons mis en place le terrain via les fichiers de données, qui nous permettent de recréer le déniveler du terrain en obtenant la longitude, latitude et l'élévation par rapport au repère initial, auquel nous avons ajouté la texture avec l'image du site. Ensuite pour affiner la modélisation nous avons ajouté différents éléments en récupérant des données des fichiers GeoJSON :

- Un chemin partant du bâtiment 333 montant sur le plateau avec les différents arrêts
- La ligne du RER B ainsi que les routes afin de mieux les distinguer
- Les bâtiments ont été modélisé en 3D et colorés par secteur

II. Analyse du problème

Pour modéliser le site de l'Université en 3D cela nécessite une implémentation rigoureuse. La difficulté résidait dans la manipulation du langage, et les potentielles erreurs dans l'affichage, il fallait donc comprendre pour pouvoir ajuster le résultat afin qu'il soit correct. Pour cela, nous avons utilisé un fichier par classe en plus du fichier principal, ce qui permet une lecture plus claire et évite les erreurs. Il y a donc un fichier principal, un fichier pour la caméra, un fichier pour l'espace de travail, un fichier Map3D pour les repères déjà donné, un fichier pour le déniveler du terrain ainsi que la texture, un fichier pour le chemin, un fichier pour le RER, un fichier pour les routes et ainsi qu'un fichier pour les bâtiments. Chaque fichier contient une classe, avec des fonctions update() afin de pouvoir les afficher ou non. Dans le fichier main on crée des objets issu de ces classes dans le setup() puis on les affiche grâce à la fonction draw(). Pour pouvoir afficher ce que l'on souhaite une fonction keyPressed() permet d'enlever ou de rajouter des éléments. Les fonctions mouseWheel() ainsi que mouseDragged() permettent de bouger et d'ajuster la caméra.

III. Interface utilisateur et évènements clavier

Affichage des objets

A Affiche ou non le terrain

B Affiche ou non les bâtiments

L Affiche ou non l'éclairage personnalisé

R Affiche ou non les voies routières et ferroviaires

W Affiche ou non l'espace de travail

X Affiche ou non le tracé GPS

Gestion de la caméra

Dézoom

Déplace à gauche la caméra

Déplace à droite la caméra

Zoom

De la même manière, on peut déplacer la caméra en cliquant sur la molette et en déplaçant en même temps la souris

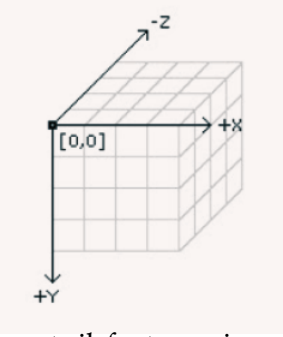
Un clic gauche permet d'afficher la description du point de cheminement concerné

IV. Développement du projet

a. Mise en place de l'espace de travail

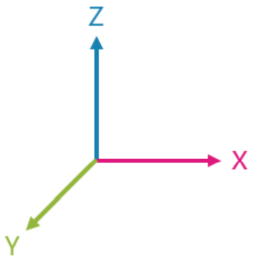
L'une des premières étapes de ce projet est de mettre en place notre espace de travail. Pour cela, nous avons défini notre propre repère cartésien, ainsi qu'une grille pour le plan de notre sol et d'autre part nous avons implémenté le positionnement de la caméra. Afin de mieux organiser notre travail, ces différents objets ont été répartis dans une classe WorkSpace et Camera.

1.



1. Premièrement, il faut savoir que Processing utilise un repère par défaut main gauche avec les Y positifs vers le bas comme on peut le voir ici.

2.



2. Or, pour notre projet, nous allons plutôt vouloir utiliser le repère cartésien 3D suivant, que l'on appellera par la suite gizmo.

Pour chacun des objets que nous allons créer dans ce projet, la marche à suivre restera la même. On commence par déclarer l'objet avec PShape nomObjet, puis on le modélisera dans le constructeur de la classe et enfin on l'affichera à l'aide de la méthode update() qui utilisera la commande shape(nomObjet). Pour la construction de notre gizmo, nous utilisons le type LINES pour nous permettre de créer une ligne entre les deux sommets de coordonnées que l'on a choisi. Et voici le résultat.

Comme dit précédemment, nous allons réaliser les mêmes étapes que pour l'objet gizmo mais pour l'objet grid qui est la grille du sol. Pour sa construction, nous utilisons le type QUADS qui prends quatre sommets de coordonnées : s1 = (x , y , z)

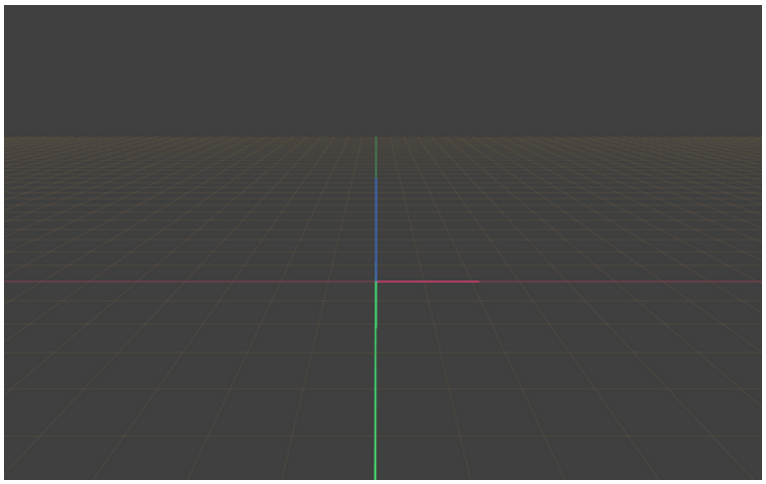
s2 = (x , y+1 , z)

s3 = (x+1 , y+1 , z)

s4 = (x+1 , y , z)

pour former un carré. Comme nous voulons une grille composée de 100*100 (= size*size) dalles, nous commençons par faire une boucle pour i allant de -50 à 50 qui va créer une première colonne de dalles centrée en (0,0,0) puis une deuxième boucle pour j allant de -50 à 50 qui va créer pour chaque colonne une ligne de 100 dalles.

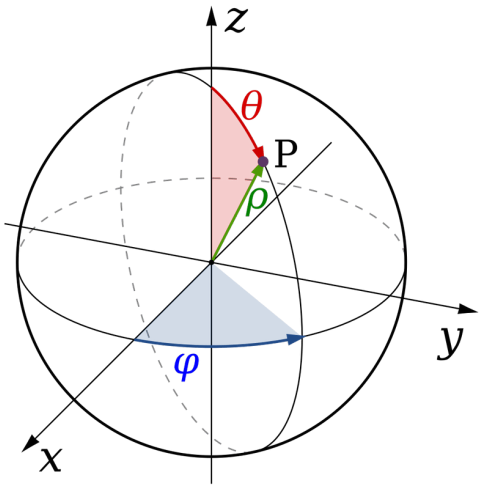
```
for(int i = -size/2; i<=size/2;i++){
  for(int j = -size/2;j <=size/2;j++) {
    int x1 = size_slab*j;
    int y1 = size_slab*i;
    int x2 = size_slab*(j+1);
    int y2 = size_slab*(i+1);
    this.grid.vertex(x1,y1,0);
    this.grid.vertex(x1,y2,0);
    this.grid.vertex(x2,y2,0);
    this.grid.vertex(x2,y1,0);
  }
}
```



Il ne nous reste plus qu'à créer nos objets dans le setup() avec la commande this.gizmo = new WorkSpace(100) et de les afficher dans draw avec this.gizmo.update(). Et voici le résultat !

La seconde étape de cette partie est d'implémenter une caméra mobile capable de pivoter autour de l'origine en (0,0,0) de notre nouveau repère 3D. Il faut savoir que la caméra est positionnée à la surface d'une sphère virtuelle d'origine (0,0,0). Par conséquent, ses coordonnées sont exprimées en coordonnées sphériques, c'est-à-dire en fonction du rayon de la sphère et des angles colatitude et longitude.

3.



3. *Illustration de la convention de l'article. La position du point P est définie par la distance ρ et par les angles θ (colatitude) et φ (longitude).*

Pour faire pivoter la caméra, nous avons donc créé des méthodes adjustColatitude(float delta), adjustLongitude(float delta) et adjustRadius(float offset) qui ajoute ou retire le nombre offset (resp. delta) aux variables concernées. De cette manière, en initialisant delta à PI/36 (=5°) et offset à 100 (=100 mètres), on peut grâce aux évènements clavier déplacer la caméra autour de l'origine (voir section évènement clavier).

```
public void adjustRadius(float offset){
  if (this.radius+offset >= width*0.5 && this.radius+offset <= width*3.0){
    this.radius += offset;
    this.cartesian();
  }
}

public void adjustLongitude(float delta){
  if (this.longitude+delta >= -3*(PI/2) && this.longitude+delta <= PI/2){
    this.longitude += delta;
    this.cartesian();
  }
}

public void adjustColatitude(float delta){
  if (this.colatitude+delta >= epsilon && this.colatitude+delta <= PI/2){
    this.colatitude += delta;
    this.cartesian();
  }
}
```

De plus, nous avons dû convertir les coordonnées originelles cartésiennes de la caméra en coordonnées sphériques.

x = 0	=>	rayon =500*sqrt(29)
y = 2500	=>	longitude = -PI/2
z = 1000	=>	colatitude = 0.38*PI

b. Modélisation 3D du terrain et application de texture à partir d'image satellite

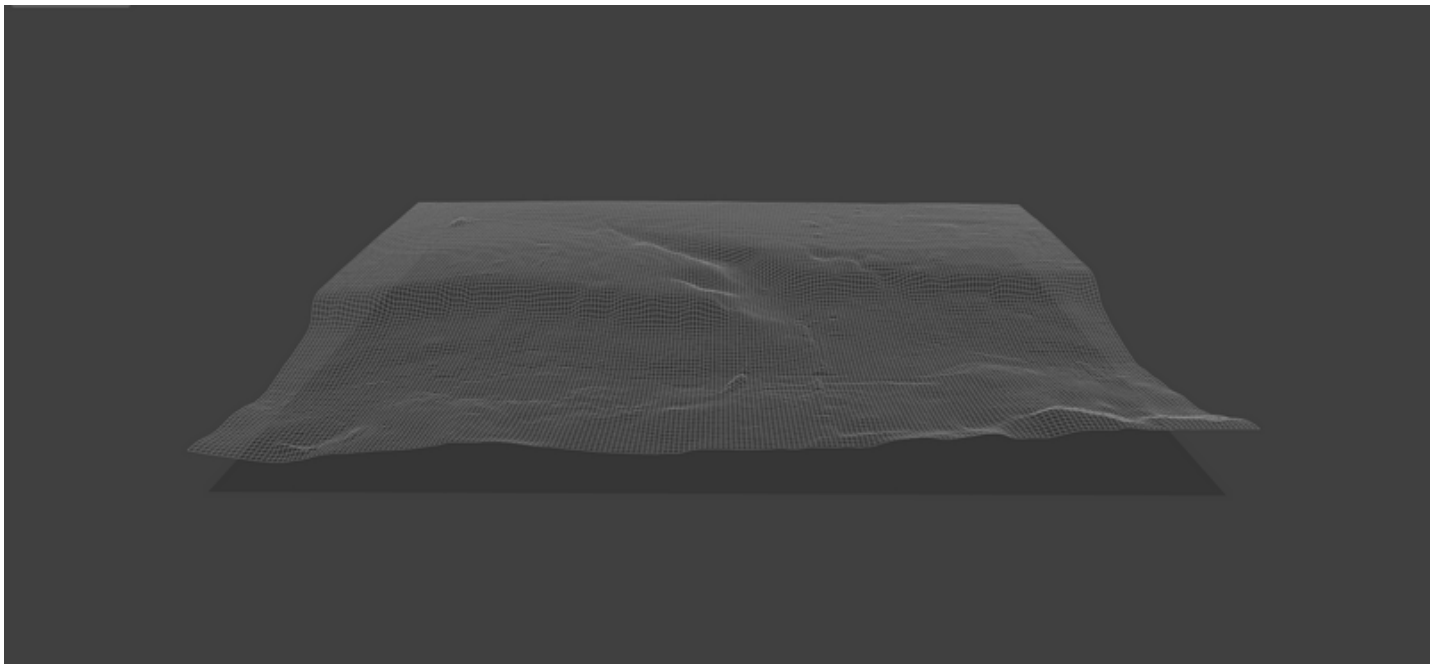
Maintenant que nous avons mis en place notre plan de travail en 3D nous allons pouvoir passer à la modélisation du terrain de Paris-Saclay limité par une projection de 5000 par 3000 mètres. Pour cela nous modéliserons dans une classe Land, les objets comme l'ombre du terrain, la grille du terrain avec ses différents dénivelés auquel on lui appliquera une texture. À cet effet, une classe Map3D nous est fournie afin de pouvoir effectuer les différents changements de systèmes de coordonnées dont nous aurons besoin.

Premièrement, il est très facile de modéliser l'ombre du terrain par une PShape de type QUADS qui va former un rectangle. Il suffit de fournir quatre sommets qui correspondent aux coordonnées des limites de l'objet map de type Map3D.

```
this.shadow.vertex(-w/2,+h/2,z_shadow);
this.shadow.vertex(-w/2,-h/2,z_shadow);
this.shadow.vertex(+w/2,-h/2,z_shadow);
this.shadow.vertex(+w/2,+h/2,z_shadow);
```

Pour la création du maillage 3D du terrain en fil de fer, on réutilise la même méthode que pour l'objet grid de la partie A. La seule différence, est que l'on doit récupérer les coordonnées de l'élévation au point p pour l'ajouter lorsque l'on définit les sommets à tracer. Pour ce faire, on instancie un objet de la classe ObjectPoint de Map3D et on récupère la coordonnée z avec nomObjectPoint.z.

```
for(float i = -nbTileHeight/2; i < nbTileHeight/2; i++){
  for(float j = -nbTileWidth/2; j < nbTileWidth/2; j++) {
    float x1 = tileSize*j;
    float y1 = tileSize*i;
    float x2 = tileSize*(j+1);
    float y2 = tileSize*(i+1);
    //creates four processing coordinates points which permits the retrieval of the elevation (z)
    Map3D.ObjectPoint p1 = this.map.new ObjectPoint(x1,y1);
    Map3D.ObjectPoint p2 = this.map.new ObjectPoint(x1,y2);
    Map3D.ObjectPoint p3 = this.map.new ObjectPoint(x2,y2);
    Map3D.ObjectPoint p4 = this.map.new ObjectPoint(x2,y1);
    this.wireFrame.vertex(p1.x,p1.y,p1.z); //vertice 1 ( x   , y   , z )
    this.wireFrame.vertex(p2.x,p2.y,p2.z); //vertice 2 ( x   , y+1 , z )
    this.wireFrame.vertex(p3.x,p3.y,p3.z); //vertice 3 ( x+1 , y+1 , z )
    this.wireFrame.vertex(p4.x,p4.y,p4.z); //vertice 4 ( x+1 , y   , z )
  }
}
```

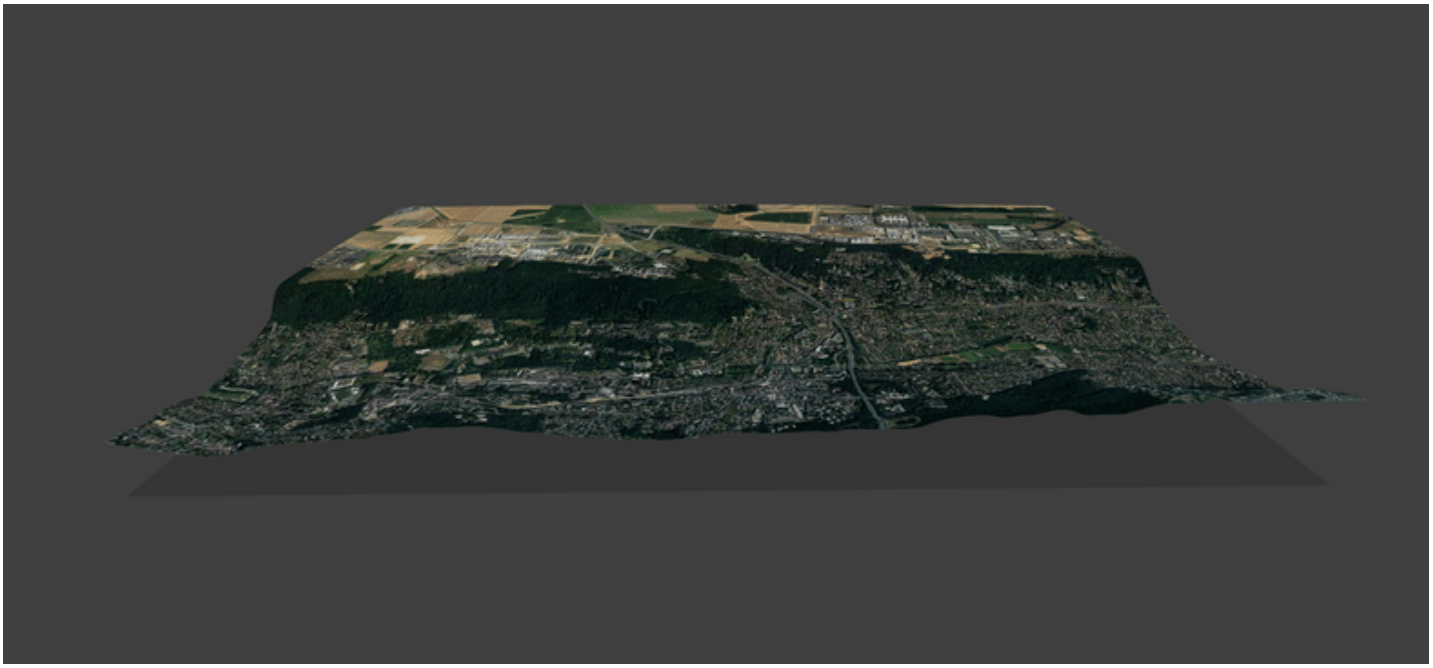


Maintenant que nous venons de finir de tracer le maillage 3D de la map on va pouvoir lui appliquer une texture en créant un nouvel objet nommé satellite (vue satellite de la map) qui reprend le code pour le maillage à quelques changements près. Pour pouvoir lui appliquer une texture on commence par écrire cette commande

```
this.satellite.texture(uvmap);
```

Pour ajouter la texture au maillage, nous allons ajouter des coordonnées u et v aux sommets qui correspondent aux points de la texture que l'on veut associer aux points de maillage correspondant. Et voici le résultat obtenu pour cette partie !

```
//associates the points of coordinates (u,v) of the texture to the points of coordinates (x,y,z) of the wireframe
this.satellite.vertex(p1.x, p1.y, p1.z, u1, v1); //vertice 1 ( x1   , y1   , z1 , x2   , y2   )
this.satellite.vertex(p2.x, p2.y, p2.z, u1, v2); //vertice 2 ( x1   , y1+1 , z1 , x2   , y2+1 )
this.satellite.vertex(p3.x, p3.y, p3.z, u2, v2); //vertice 3 ( x1+1 , y1+1 , z1 , x2+1 , y2+1 )
this.satellite.vertex(p4.x, p4.y, p4.z, u2, v1); //vertice 4 ( x1+1 , y1   , z1 , x2+1 , y2   )
```



c. Visualisation de données spécifiques

À présent que nous avons notre terrain, nous allons pouvoir afficher notre premier tracé GPS commençant au bâtiment PUIO sur le plateau et finissant au parking du bâtiment 333. Pour cela nous allons utiliser un fichier GPX qu'on convertira au format GeoJSON. De cette manière, il sera assez facile pour nous de créer nos trois objets track, post et thumbtack qui formeront la trace et les points de cheminements du chemin, à partir des points géographiques donnés par le fichier.

Pour modéliser la trace, on commence par récupérer les coordonnées d'un premier point qu'on va stocker dans une variable et que l'on va convertir en ObjectPoint. Par la suite, on trace un sommet avec ces coordonnées qui grâce au type LINE_STRIP sera relié par une ligne avec le prochain point que l'on va récupérer.

```
Map3D.GeoPoint Mp1 = this.map.new GeoPoint(point.getDouble(0), point.getDouble(1));
Mp1.elevation += 1.5d;
Map3D.ObjectPoint Obj1 = this.map.new ObjectPoint(Mp1);
this.track.vertex(Obj1.x, Obj1.y, Obj1.z);
```

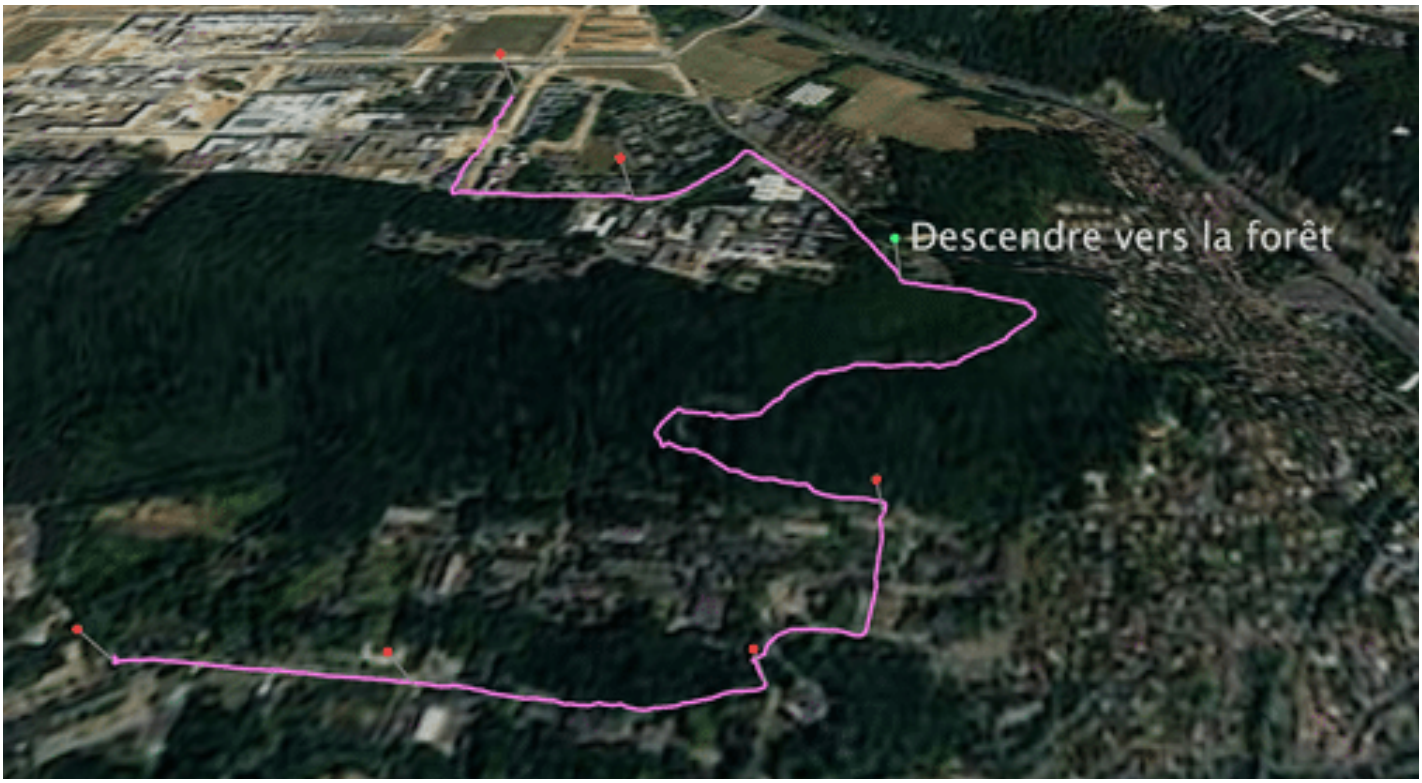
D'autre part, nous voulons tracer des petits bâtonnets avec une punaise rouge afin d'identifier des points de cheminements tout du long de la trace. Pour cela on procède de la même manière qu'avant, à la seule différence que nous allons récupérer les coordonnées de points de cheminements et pas de la trace. Puis, il suffit de tracer une ligne pour chaque coordonnées récupérées avec seulement la hauteur qui varie et d'y placer à cette hauteur un petit point (=thumbtack).

```
Map3D.MapPoint Mp = this.map.new MapPoint(this.map.new GeoPoint(point.getDouble(0), point.getDouble(1)));
Map3D.ObjectPoint Obj = this.map.new ObjectPoint(Mp);
this.posts.vertex(Obj.x, Obj.y, Obj.z);
this.posts.vertex(Obj.x, Obj.y, Obj.z+70);
this.thumbtacks.vertex(Obj.x, Obj.y, Obj.z+70);
```

Additionnellement, il serait intéressant de faire en sorte que la description des points de cheminement du tracé puissent s'afficher lorsque l'utilisateur fait un clic gauche sur une des punaises. Pour cela, nous avons fait une méthode clic qui permet de sélectionner une punaise lors d'un clic gauche et donc de changer sa couleur. Pour afficher la description il suffit alors de gérer son affichage dans la fonction update() en récupérant ses coordonnées et le numéro du point avec des variables globales.

```
public void clic(float mouseX,float mouseY) {
  for (int v = 0; v< this.thumbtacks.getVertexCount();v++){
    this.hit = new PVector();
    this.hit = this.thumbtacks.getVertex(v,hit);
    float distThumbtacks = dist(mouseX,mouseY,screenX(hit.x,hit.y,hit.z),screenY(hit.x,hit.y,hit.z));
    if (distThumbtacks < 10) {
      this.thumbtacks.setStroke(v, 0xFF3FFF7F);
      desc = v;
      this.hitBis = new PVector();
      hitBis = this.hit;
    } else {
      this.thumbtacks.setStroke(v, 0xFFFF3F3F);
    }
  }
}
```

Et voici le résultat en image !



d. Tracé de routes

Une fois familiarisées avec l'utilisation de fichier GeoJSON, nous allons pouvoir utiliser ces mêmes fichiers pour créer les voies routières, ferroviaires et pédestres de tout Paris-Saclay.

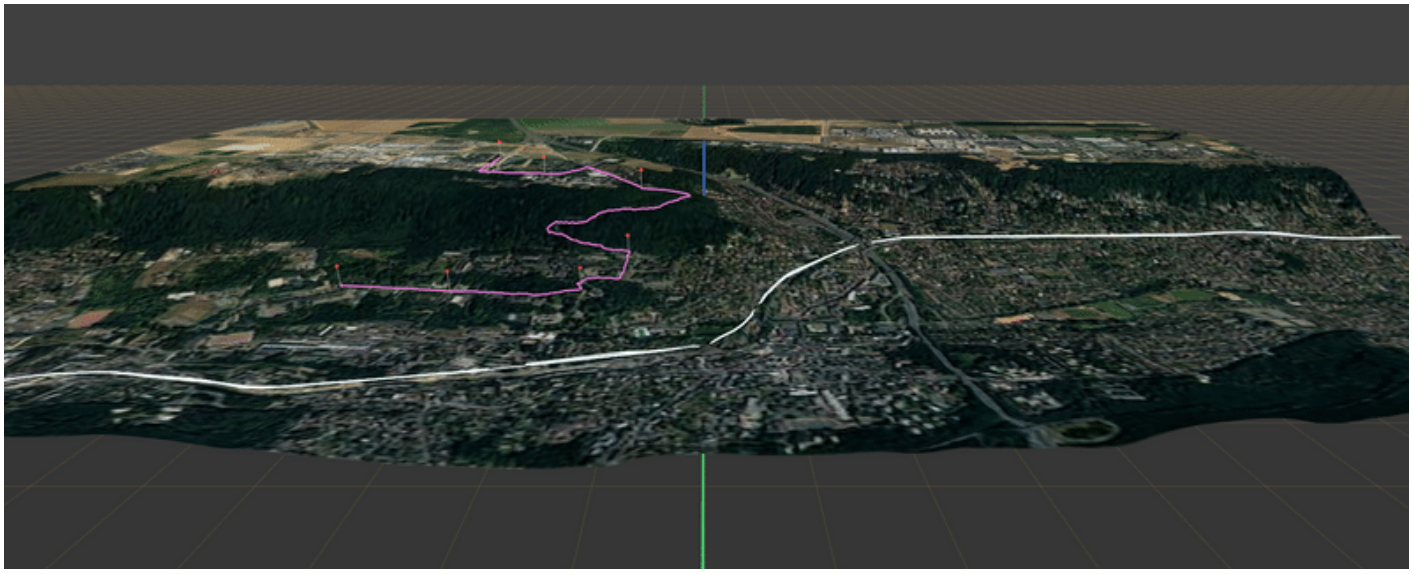
Dans un premier temps, nous allons modéliser un objet qui représentera la ligne du RER B dans une classe nommée Railways et qui sera de type GROUP. Autrement dit, nous allons créer un deuxième objet railtrack qui sera ensuite ajouté à l'objet railways via ces commandes :

```
this.railways = createShape(GROUP);

this.railways.addChild(this.railtrack);
```

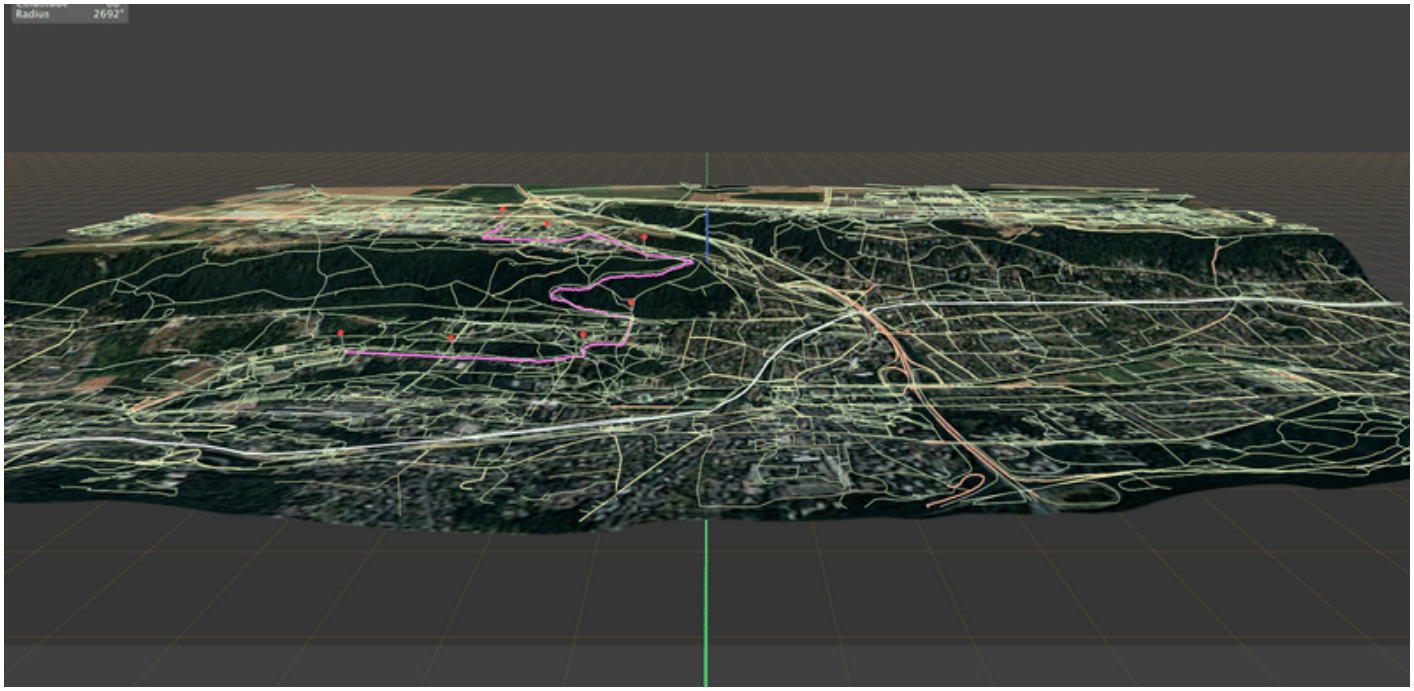
Pour la création de l'objet railtrack qui représente la trace du RER B on procède de la même manière que précédemment pour tracé GPS, c'est-à-dire qu'on commence par récupérer les coordonnées des points de traces. Comme le RER B à deux railles nous allons tracer cela en utilisant des vecteurs normaux aux points qu'on ajoutera ou déduira en fonction du sens voulu. Comme certains points peuvent être en dehors de la limite de notre terrain nous vérifions avant de le tracer que le point se trouve bien à l'intérieur avec nomPoint.inside(). De plus, pour des soucis d'élévation certains tronçons ne s'affichent par correctement et c'est pour cela qu'on la réhausse d'une valeur de 7.5d pour être sûr que ceux-ci soient visibles.

```
Map3D.GeoPoint Mp1 = this.map.new GeoPoint(point.getDouble(0), point.getDouble(1));
Map3D.GeoPoint Mp2 = this.map.new GeoPoint(point.getDouble(0), point.getDouble(1));

if (Mp1.inside()){
  Mp1.elevation += 7.5d;
  Map3D.ObjectPoint Obj1 = this.map.new ObjectPoint(Mp1);
  Map3D.ObjectPoint Obj2 = this.map.new ObjectPoint(Mp2);
  PVector Va = new PVector(Obj1.y - Obj2.y, Obj2.x - Obj1.x).normalize().mult(this.lineWidth/2.0f);
  this.railtrack.vertex(Obj1.x - Va.x ,Obj1.y - Va.y,Obj1.z);
  this.railtrack.vertex(Obj1.x + Va.x,Obj1.y + Va.y,Obj1.z);

-
```

Nous venons de tracer la ligne du RER B !



Pour les routes, il s'agit d'une généralisation du cas du RER B. Il suffit de copier-coller la classe Railways dans une classe Roads qui à partir d'un fichier roads.GeoJSON va créer toutes les routes de Paris-Saclay. La seule différence notable est que nous modifions la largeur et la couleur de la route en fonction de son type.
Nous venons de finir cette partie du projet ! Nous pouvons voir toutes les voies de Paris-Saclay !



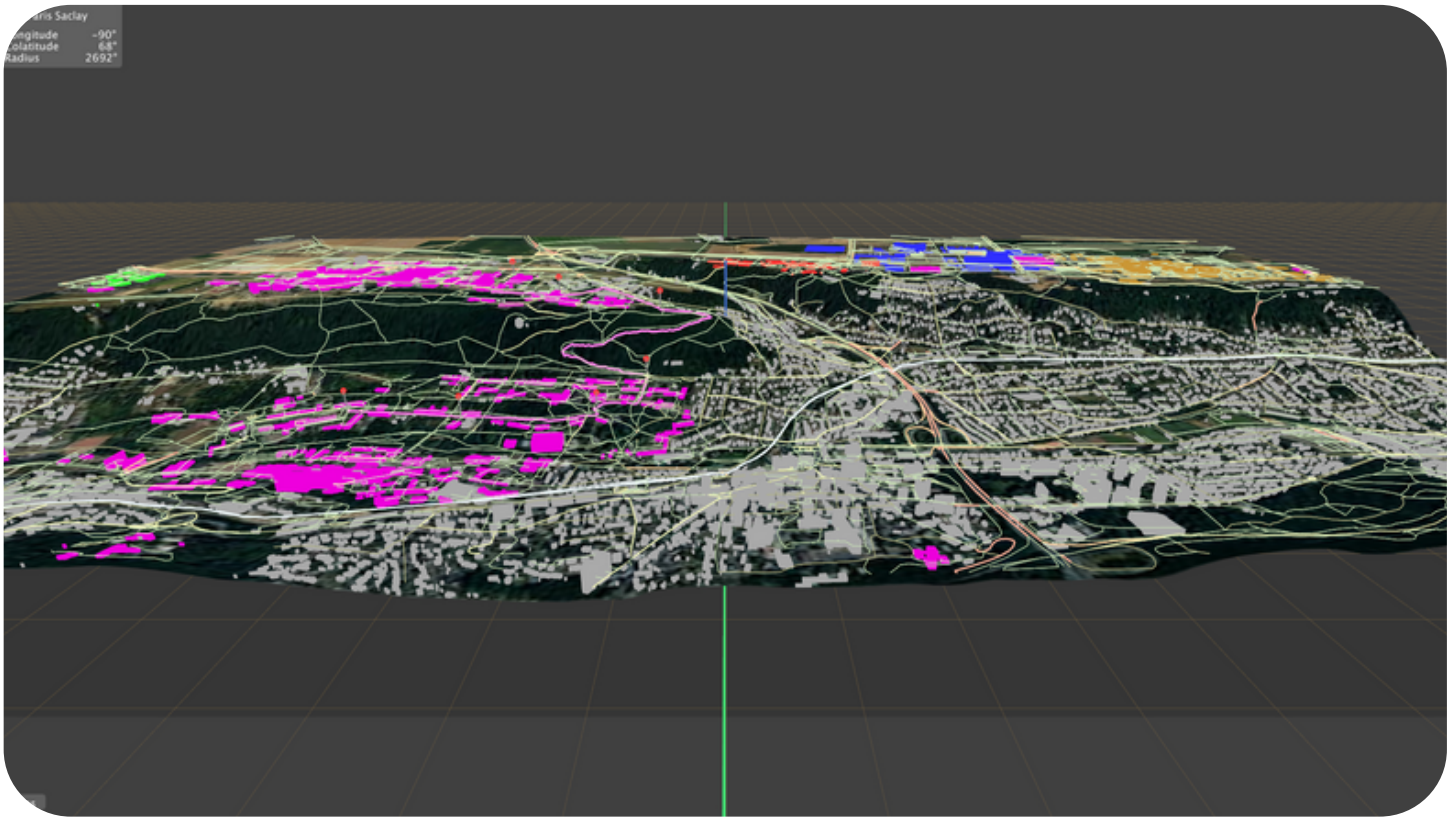
e. Tracé de bâtiments

Notre dernière modélisation sera les bâtiments, toujours à l'aide de nos fichiers GeoJSON. Pour commencer on implémente l'objet buildings qui est un tableau contenant des PShape building de type GROUP. Chaque PShape building contient les bâtiments d'une zone particulière telle que le plateau, le CEA… Pour construire ces différents building, il faut construire un à un chaque bâtiment. Pour chacun d'entre eux on construit donc un objet roof de type POLYGON ainsi que des objets walls de type QUAD_STRIP, on construit les quatre murs et le toit en récupérant les coordonnées dans le fichier GeoJSON et en choisissant la hauteur des bâtiments. Encore une fois, avant de les construire on vérifie que les points sont bien à l'intérieur de notre terrain grâce à nomPoint.inside(). Une fois le bâtiment tracé on l'ajoute à building, ensuite lorsque tous les bâtiments de la zone sont créés on ajoute building au tableau buildings. On répète l'opération pour toutes les zones et on obtient la totalité du territoire.

```
        private ArrayList<PShape> buildings = new ArrayList<PShape>();

if (Gp1.inside() && Gp2.inside()) {
  Map3D.ObjectPoint Obj1 = this.map.new ObjectPoint(Gp1);
  Map3D.ObjectPoint Obj2 = this.map.new ObjectPoint(Gp2);
  walls.vertex(Obj1.x,Obj1.y,Obj1.z);
  walls.vertex(Obj1.x,Obj1.y,Obj1.z + top);
  walls.vertex(Obj2.x,Obj2.y,Obj2.z);
  walls.vertex(Obj2.x,Obj2.y,Obj2.z + top);
  roof.vertex(Obj1.x,Obj1.y,Obj1.z + top);
}
```

```
                    this.buildings.add("buildings_city.geojson",0xFFaaaaaa);
                    this.buildings.add("buildings_IPP.geojson",0xFFCB9837);
                    this.buildings.add("buildings_EDF_Danone.geojson",0xFF3030FF);
                    this.buildings.add("buildings_CEA_algorithmes.geojson",0xFF30FF30);
                    this.buildings.add("buildings_Thales.geojson",0xFFFF3030);
                    this.buildings.add("buildings_Paris_Saclay.geojson",0xFFee00dd);
```

Nous avons construit les bâtiments !



V. Résultats, conclusions et perspectives

Nous sommes arrivées jusqu'à la fin de la fiche P05, c'est-à-dire les bâtiments. Nous obtenons comme résultat, une modélisation du terrain de l'université avec les bâtiments, les routes, la ligne du RER ainsi qu'un chemin. Pour conclure, nous avons avancé pas à pas dans un projet difficile qui se mettait en place petit à petit. Chaque étape permettant de modéliser un nouvel élément, ce fut gratifiant de voir cette modélisation évoluer un peu plus à chaque fois, le résultat se concrétisant directement. Afin de faciliter l'usage à l'exécution, nous aurions pu ajouter un panneau récapitulatif de toutes les commandes, pour que l'utilisateur profite pleinement de la modélisation. Pour rendre ce projet encore plus réaliste, nous aurions pu rajouter un bloc de terre comprenant les différentes couches en dessous de la modélisation, afin de le représenter comme une coupe de terrain directement découpée de la terre. Il serait également intéressant de voir si des bases de données renseignent les cours d'eau et les zones de végétation pour les modéliser en 3D également ou bien encore la faune et la flore présente sur le territoire, la modélisation serait de plus en plus complète. Ces dernières pourraient être modélisées par deux icônes, un pour la flore et un pour la faune, ainsi lorsque l'utilisateur cliquerait dessus il aurait accès à la liste de chacun. De plus, la mise en relief d'un élément ainsi que l'affichage des informations de celui-ci lorsque l'on passe la souris dessus pourrait rendre la modélisation ludique pour l'apprentissage notamment. Pour finir, ce projet nous a permis d'imaginer l'étendue de ce que l'on peut faire grâce à l'informatique graphique, ce qui est impressionnant et inspirant pour la poursuite de nos études.