4) Affichage d'un tracé GPS

4.1) Les fichiers GPX

La norme *GPs eXchange format*[9] décrit un format de fichier permettant l'échange de coordonnées GPS. Elle définit une succession de points géographiques utilisables sous forme de point de cheminement (*waypoint*) formant une trace (*track*).

Ces fichiers sont exportés depuis un boitier GPS, ou plus simplement depuis une application d'un smartphone telle que *OsmAnd* sur Android ou *GPX Tracker* sur iOS.

4.1.1) Conversion au format GeoJSON

Vous devez récupérer le fichier « *trail.gpx* » qui contient un tracé d'exemple, qui part du bâtiment PUIO sur le plateau et se termine au parking du Bâtiment 333.

Afin de faciliter la récupération des données dans Processing, le fichier GPX est converti au format *GeoJSON*. Vous pouvez par exemple ouvrir le fichier GPX sur le site *geojson.io* via le menu « *Open* » :

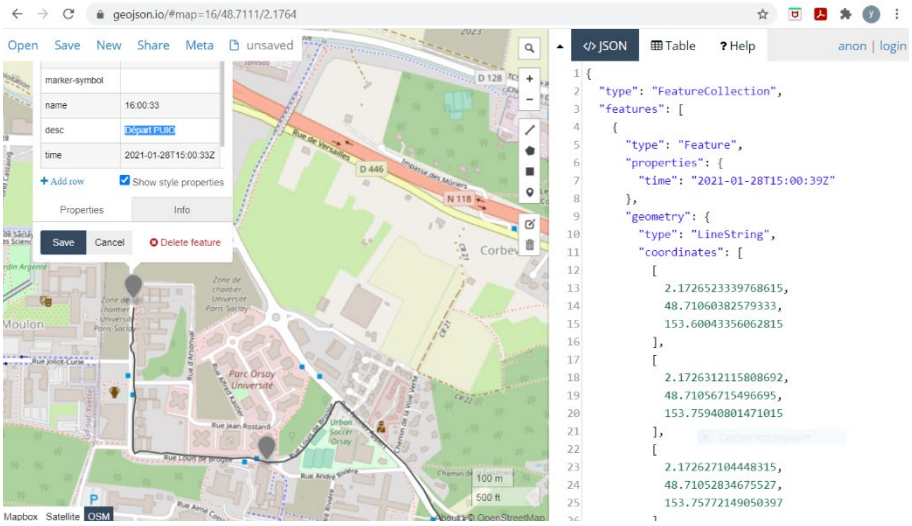

Figure 15 - Import GPX dans geojson.io

Le format *GeoJSON* (JSON géographique)[10] est un format ouvert décrivant un ensemble de données géospatiales simples utilisant la norme JSON (JavaScript Object Notation). Ce format permet de décrire des ensembles de données spatiales de type points, lignes ou polygones, ainsi que des attributs d'informations associées.

[9] Voir également : Wikipédia - GPX (format de fichier)
[10] Voir également : Wikipédia - GeoJSON

Le menu « *Save* » permet d'exporter le fichier converti.

4.1.2) Parcours d'un fichier GeoJSON

Un fichier GeoJSON contient obligatoirement un objet *features* de type *featureCollection*.

Ensuite chaque fonctionnalité est décrite dans un objet *feature* qui comprend lui-même deux parties optionnelles : Un sous objet *geometry* qui décrit les données spatiales, et un sous objet *properties* qui contient les informations associées.

L'algorithme de lecture du fichier GPX converti est le suivant :

```
String fileName = "trail.geojson";

// Check ressources
File ressource = dataFile(fileName);
if (!ressource.exists() || ressource.isDirectory()) {
  println("ERROR: GeoJSON file " + fileName + " not found.");
  return;
}

// Load geojson and check features collection
JSONObject geojson = loadJSONObject(fileName);
if (!geojson.hasKey("type")) {
  println("WARNING: Invalid GeoJSON file.");
  return;
} else if (!"FeatureCollection".equals(geojson.getString("type", "undefined"))) {
  println("WARNING: GeoJSON file doesn't contain features collection.");
  return;
}

// Parse features
JSONArray features =  geojson.getJSONArray("features");
if (features == null) {
  println("WARNING: GeoJSON file doesn't contain any feature.");
  return;
}
for (int f=0; f<features.size(); f++) {

  JSONObject feature = features.getJSONObject(f);
  if (!feature.hasKey("geometry"))
    break;
  JSONObject geometry = feature.getJSONObject("geometry");
  switch (geometry.getString("type", "undefined")) {

  case "LineString":

    // GPX Track
    JSONArray coordinates = geometry.getJSONArray("coordinates");
    if (coordinates != null)
      for (int p=0; p < coordinates.size(); p++) {
        JSONArray point = coordinates.getJSONArray(p);
        println("Track ", p, point.getDouble(0), point.getDouble(1));
      }
    break;

  case "Point":

    // GPX WayPoint
    if (geometry.hasKey("coordinates")) {
```

```
      JSONArray point = geometry.getJSONArray("coordinates");
      String description = "Pas d'information.";
      if (feature.hasKey("properties")) {
        description = feature.getJSONObject("properties").getString("desc",
description);
      }
      println("WayPoint", point.getDouble(0), point.getDouble(1), description);
    }
    break;

  default:
    println("WARNING: GeoJSON '" + geometry.getString("type", "undefined") + "' geometry
type not handled.");
    break;
  }
}
```

Cet algorithme parcourt la trace GPS et les points de cheminements marqués :

```
Track  0 2.1726523339768615 48.71060382579333
Track  1 2.1726312115808692 48.71056715496695
…
Track  440 2.168940072881215 48.70249622040307
Track  441 2.168926829474204 48.702502129644806
WayPoint 2.1726349522673973 48.71073399338086 P.U.I.O
WayPoint 2.176210852338008 48.7078706918397 Arrêt de bus
WayPoint 2.1815433634620605 48.706105949905435 Descendre vers la forêt
WayPoint 2.1812395937761115 48.703657952182645 Intersection dangereuse
WayPoint 2.179579716447848 48.70174834999577 A droite avant le château
WayPoint 2.173922143837632 48.70202805323668 Feu tricolore
WayPoint 2.168928460910572 48.70246494789282 Parking Bat. 333
```

4.2) Réalisation du tracé

Vous devez créer une classe *Gpx* dans un nouvel onglet *Gpx*.

Vous créerez un objet *Gpx gpx;* global dans le programme principal, qui sera initialisé dans la fonction setup par *this.gpx = new Gpx(this.map, "trail.geojson");*

Le constructeur de la classe va créer trois formes *PShape*.

Tout d'abord, une *PShape track* qui contiendra la succession des points du tracé. Grâce à l'objet *Map3D* passé en paramètres, les points à afficher seront obtenus en utilisant des objets *geoPoint* convertis en *objectPoint*.

NB : Bien que le format GPX puisse fournir une altitude, il est recommandé de laisser la classe Map3D retrouver les élévations fournies par l'IGN ; les 2 systèmes n'utilisant pas la même référence d'altimétrie, cela évitera que votre tracé ne « disparaisse » sous votre terrain.

Pour matérialiser des « punaises » représentant les *WayPoint* sur le tracé à l'écran, vous créerez une *PShape posts* de type *LINES* pour les « poteaux » et une *PShape thumbtacks* de type

POINTS pour les « têtes d'épingles » [11] (*cf. photo ci-après*).

NB : Bien que positionnées dans un espace 3D, les formes de type LINES *&* POINTS *sont représentées spécifiquement par Processing. Les tracés sont orientés face à la caméra et leur taille, spécifiée par strokeWeight, reste constante indépendamment de la distance de la caméra.*

Comme pour les classes précédentes, une méthode *update* prendra en charge l'affichage des formes durant la fonction *draw*, et une méthode *toggle* permettra de gérer leurs visibilités à l'aide de la touche « *X* » du clavier.

Félicitations, vous avez réalisé votre premier tracé GPS en 3D :

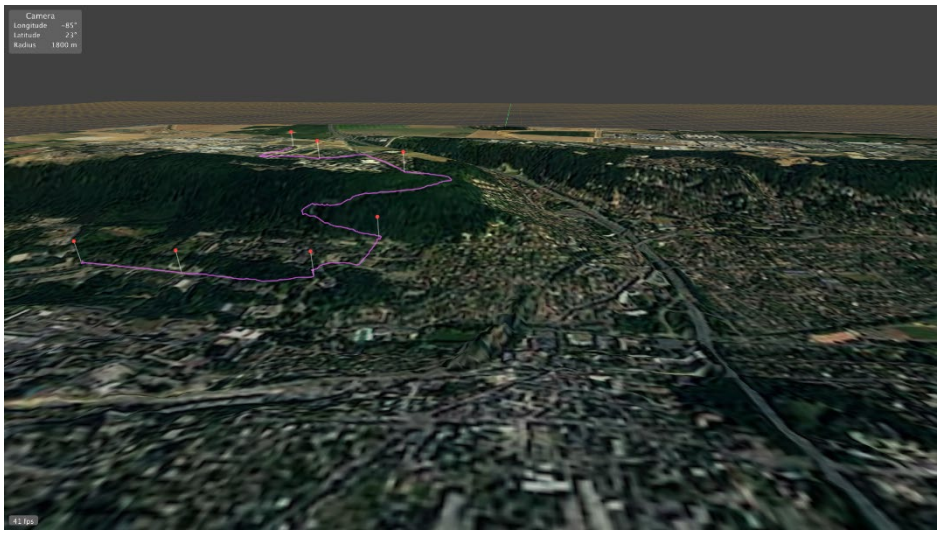

Figure 16 - Tracé GPS

4.3) Sélection utilisateur

Vous allez proposer à l'utilisateur de sélectionner par un clic gauche de la souris une des punaises rouges du tracé, afin d'afficher la description du *WayPoint* sélectionné.

4.3.1) Sélection d'un point du tracé

Dans le programme principal, vous appellerez une méthode *clic()* de la classe *Gpx* :

```
void mousePressed() {
  if (mouseButton == LEFT)
    this.gpx.clic(mouseX, mouseY);
}
```

Cette méthode devra comparer à l'aide de la fonction *dist()* de Processing la position de la souris passée (*mouseX*, *mouseY*) avec les positions à l'écran des sommets d'indice *{v = 0 ; v < this.thumbtacks.getVertexCount() ; v++}* des punaises, données par les fonctions *screenX()* et

[11] Voir les différents types dans la documentation Processing : beginShape()

screenY(). Vous utiliserez à cette fin la fonction *this.thumbtacks.getVertex(v,hit)* pour récupérer dans un objet *PVector hit* les coordonnées d'un sommet précédemment créé.

Quand une punaise d'indice *s* est sélectionnée, sa couleur devra être modifiée via *this.thumbtacks.setStroke(s, 0xFF3FFF7F)*. Les punaises d'indice p non sélectionnées reprennent leur couleur d'origine via *this.thumbtacks.setStroke(p, 0xFFFF3F3F)*.

4.3.2) Affichage de texte

Comme vous l'avez déjà vu pour l'affichage d'informations dans la classe *Hud*, le tracé de textes s'effectue indépendamment des formes.

L'exemple suivant montre comment afficher une *description* à une position de punaise déterminée par une variable *PVector hit* :

```
pushMatrix();
lights();
fill(0xFFFFFFFF);
translate(hit.x, hit.y, hit.z + 10.0f);
rotateZ(-camera.longitude-HALF_PI);
rotateX(-camera.colatitude);
g.hint(PConstants.DISABLE_DEPTH_TEST);
textMode(SHAPE);
textSize(48);
textAlign(LEFT, CENTER);
text(description, 0, 0);
g.hint(PConstants.ENABLE_DEPTH_TEST);
popMatrix();
```

La variable globale de gestion de la caméra sera passée en paramètre *camera*, afin que le texte soit toujours orienté face à l'utilisateur, indépendamment des mouvements de la caméra.

L'inhibition du test de profondeur du pipeline graphique permet au texte de toujours s'afficher, y compris en deçà d'éventuelles collines.

Félicitations, vous savez désormais interagir avec l'utilisateur de votre application :

Figure 17 - Sélection d'un point de cheminement