3) Modélisation du terrain

3.1) Systèmes de coordonnées

Vous allez maintenant devoir représenter des informations cartographiques.

Les données non projetées sont fournies dans un système de coordonnées géodésiques.

- Les coordonnées (longitude, latitude) sont données en degrés décimaux.
- Les altitudes, si elles sont disponibles, sont des hauteurs par rapport à l'ellipsoïde de référence.

Les données projetées sont fournies dans un système de projection plane pour les positionner sur une carte.

- Les coordonnées (X, Y) des points sont données en mètres.
- Les élévations, si elles sont disponibles, sont des hauteurs par rapport à une référence (*généralement le niveau de la mer à marée basse*).

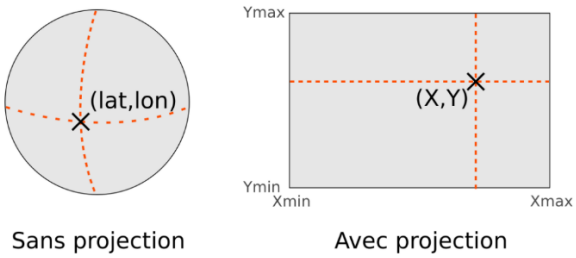

Figure 8 - Coordonnée d'un point sur le globe terrestre et dans un espace projeté

3.1.1) Limites territoriales du projet

Le territoire que vous allez représenter est limité par une projection de 5 000 x 3 000 mètres, définie par une « boîte englobante » (en anglais *bounding box*) de 4 points :

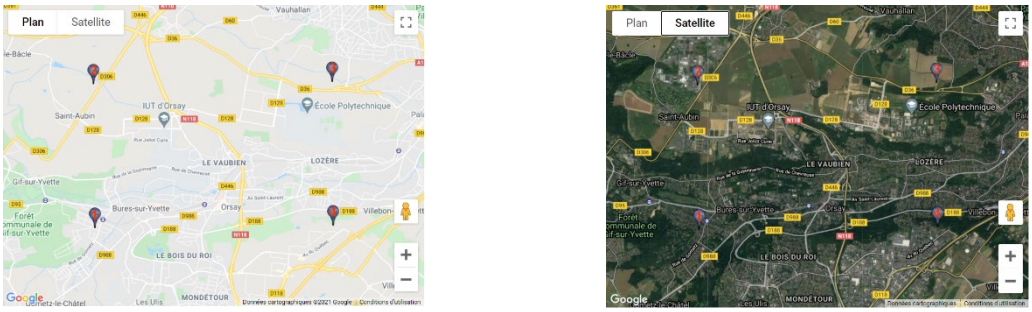

Figure 9 - Territoire Paris Saclay[7]

[7] Images obtenues via le site : GeoFree - Conversion entre Systèmes de Coordonnées GPS & SIG

Voici les coordonnées de ces 4 points dans les 3 repères que vous pourrez utiliser :

Limites	WGS84 / RGF93 (dd)		Lambert93 (m)		Objet 3D	
	Longitude	Latitude	X	Y	X	Y
1 - *Lower Left*	2.1508442	48.6931245	637500	6844000	-2500	+1500
2 - *Upper Left*	2.1504057	48.7201061	637500	6847000	-2500	-1500
3 - *Upper Right*	2.2183686	48.7205706	642500	6847000	+2500	-1500
4 - *Lower Right*	2.2187721	48.6935887	642500	6844000	+2500	+1500

3.1.2) Système de coordonnées géodésiques

Les données issues d'*Open Street Map* sont fournies dans le système de coordonnées habituel des GPS, à savoir le système WGS84.

L'unité des longitudes & latitudes est le degré décimal (dd).

Ce système est compatible avec le Réseau Géodésique Français 1993 (RGF93) utilisé par les services publics (i.e. Données INSEE).

3.1.3) Système de projection géographique

Les données d'élévation du territoire français de l'Institut Géographique National sont fournies dans le système de projection Lambert93.

L'unité des abscisses et ordonnées est le mètre (m).

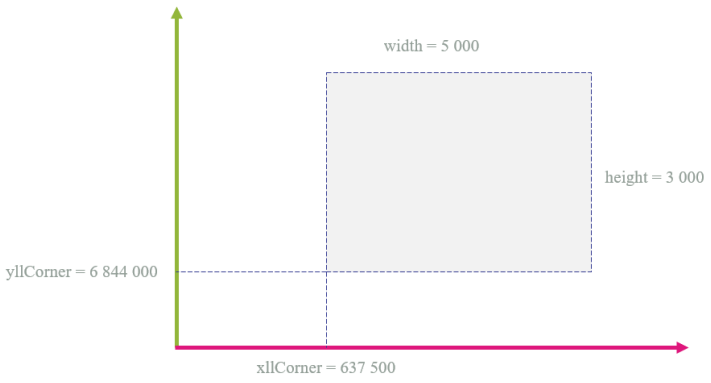

Figure 10 - Repère projeté Lambert 93

Les algorithmes de conversion entre les systèmes Lambert93 et RGF93 (équivalent à WGS84) définis par l'IGN vous sont fournis.

3.1.4) Espace de représentation d'objets 3D

Enfin, le 3ème repère que vous utiliserez sera votre repère 3D des « Terrains » dans Processing.

Il s'agit d'une translation d'origine par rapport à Lambert 93 pour centrer les formes créées en

(0,0,0), et d'une inversion de l'axe des ordonnées puisque dans Processing l'axe des ordonnées (Y+ Front) est dirigé vers vous.

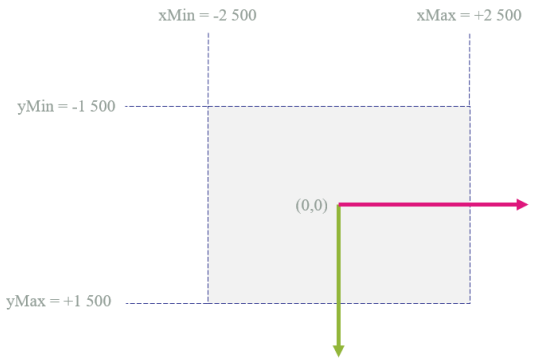

Figure 11 - Repère Objet

Sur le plan X/Y les unités équivalent aux mètres projetés. En revanche, les conversions dans ce repère Objet calculeront un facteur de mise à l'échelle des élévations, afin d'intensifier les variations d'altitudes affichées à l'écran.

3.2) Calculs de conversion

Dans le cadre de ce projet, une classe Java *Map3D* vous est fournie pour effectuer les changements de repères nécessaires. Elle inclue une classe *Geodesie* chargée de calculer les conversions entre le système géodésique WGS84 et le système projeté Lambert93.

Pour l'utiliser, vous devez ajouter le fichier *Map3D.pde* dans le dossier de votre sketch Processing.

Dans ce dossier, vous devez créer un sous-répertoire impérativement nommé « *data* », dans lequel vous stockerez le fichier de données d'élévations de l'IGN nommé « *paris_saclay.data* ».

Dans votre projet, créez une variable globale *map* de type *Map3D*.

Dans la fonction *setup*, construisez votre objet *map* en précisant le fichier d'élévations fourni (*nb : si on n'indique pas le répertoire, Processing recherche par défaut dans le dossier* « *data* ») :

```
// Load Height Map
this.map = new Map3D("paris_saclay.data");
```

Vous disposez désormais de 3 sous classes d'objets pour réaliser vos créations et conversions de points cartographiques :

- ✓ La classe *GeoPoint*, qui contient la longitude, la latitude et l'élévation dans le système GPS WGS84.
- ✓ La classe *MapPoint*, qui contient l'abscisse, l'ordonnée et l'élévation dans le système projeté Lambert 93.
- ✓ La classe *ObjectPoint*, qui contient l'abscisse, l'ordonnée et l'élévation (mise à l'échelle) dans le repère local de Processing.

Important : À chaque fois que vous créez un point dans l'un de ces repères sans indiquer l'élévation, cette information manquante sera récupérée depuis le fichier « *paris_saclay.data* ».

3.2.1) Exemple de conversion Lambert 93 en WGS 84

L'objet *MapPoint* représente une coordonnée Lambert93, tandis que l'objet *GeoPoint* représente une coordonnée WGS84.

Vous créez un objet *MapPoint* en lui fournissant des coordonnées projetées X et Y. Pour ce premier exemple, vous utiliserez les propriétés *xllCorner*, *yllCorner*, *width* & *height* qui donnent les limites du terrain en coordonnées projetées Lambert 93.

Ensuite, vous créez un objet *GeoPoint* à partir du *MapPoint* précédent.

```
// South West
Map3D.MapPoint sw = this.map.new MapPoint(
  Map3D.xllCorner,
  Map3D.yllCorner
  );
println(
  "South West :", sw,
  "\n=> ", this.map.new GeoPoint(sw)
  );

// North East
Map3D.MapPoint ne = this.map.new MapPoint(
  Map3D.xllCorner + Map3D.width,
  Map3D.yllCorner + Map3D.height
  );
println(
  "North East :", ne,
  "\n=> ", this.map.new GeoPoint(ne)
  );
```

Vos coordonnées géodésiques sont disponibles :

```
South West : xm = 637500.0, ym = 6844000.0, em = 97.4
=>  longitude = 2.150844, latitude = 48.693123, elevation = 97.4
North East : xm = 642500.0, ym = 6847000.0, em = 149.7
=>  longitude = 2.2183685, latitude = 48.72057, elevation = 149.7
```

3.2.2) Exemple de conversion Lambert 93 en coordonnées Processing

L'objet *ObjectPoint* représente une coordonnée du terrain dans l'espace de Processing :

Pour convertir les limites de votre terrain, vous pouvez procédez comme dans l'exemple précédent :

```
// South West
Map3D.MapPoint sw = this.map.new MapPoint(
  Map3D.xllCorner,
  Map3D.yllCorner
  );
println(
  "South West :", sw,
  "\n=> ", this.map.new GeoPoint(sw),
  "\n=> ", this.map.new ObjectPoint(sw)
  );

// North East
Map3D.MapPoint ne = this.map.new MapPoint(
  Map3D.xllCorner + Map3D.width,
  Map3D.yllCorner + Map3D.height
  );
println(
  "North East :", ne,
  "\n=> ", this.map.new GeoPoint(ne),
  "\n=> ", this.map.new ObjectPoint(ne)
  );
```

Vous retrouvez les coordonnées *x*, *y*, *z* des limites du terrain utilisables dans Processing :

```
South West : xm = 637500.0, ym = 6844000.0, em = 97.4
=>  longitude = 2.150844, latitude = 48.693123, elevation = 97.4
=>  x = -2500.0, y = 1500.0, z = 243.5
North East : xm = 642500.0, ym = 6847000.0, em = 149.7
=>  longitude = 2.2183685, latitude = 48.72057, elevation = 149.7
=>  x = 2500.0, y = -1500.0, z = 374.25
```

Important : Concernant les élévations en Z, elles sont mises à l'échelle (*déterminée par un facteur heightScale de 2.5*) afin que le dénivelé du terrain réel (*env. 50 mètres pour Paris Saclay*) soit accentué à l'écran. Par ailleurs, pour effectuer des tests de représentation « aplaties » en 2D, vous pourrez modifier *this.mode3D = false;* dans le constructeur de *Map3D*.

3.2.3) Exemple de conversion directe en coordonnées Processing

Vous pouvez plus simplement obtenir les limites du terrain en créant directement des objets *ObjectPoint* à partir de la largeur et de la hauteur du terrain (5000 x 3000 m), puisque dans le repère *ObjectPoint* le terrain est centré en son milieu :

```
float w = (float)Map3D.width;
float h = (float)Map3D.height;

// South West (direct)
Map3D.ObjectPoint osw = this.map.new ObjectPoint(-w/2.0f, +h/2.0f);
println("South West (direct) :", osw);

// North East (direct)
Map3D.ObjectPoint one = this.map.new ObjectPoint(+w/2.0f, -h/2.0f);
println("North East (direct) :", one);
```

Vous obtenez les résultats équivalents :

```
South West (direct) : x = -2500.0, y = 1500.0, z = 243.5
North East (direct) : x = 2500.0, y = -1500.0, z = 374.25
```

3.3) Réalisation d'un maillage du terrain en « fil de fer »

Dans un nouvel onglet Processing, vous allez créer une nouvelle classe appelée « *Land* ».

Le constructeur *Land* sera en charge de créer 2 formes.

- ✓ Une *PShape shadow;* permettra de visualiser l'ombre portée du terrain. Ici ce sera un simple rectangle de la taille du terrain, placé très légèrement sous la grille du *WorkSpace* (*pour pouvoir continuer à visualiser les graduations).*
- ✓ Une seconde *PShape wireFrame;* permettra de matérialiser le maillage 3D du terrain en « fil de fer ». À cette fin, le constructeur recevra en paramètre de construction votre objet MAP3D pour créer les points *ObjectPoint* :

```
/**
 * Returns a Land object.
 * Prepares land shadow, wireframe and textured shape
 * @param  map          Land associated elevation Map3D object
 * @return              Land object
 */
Land(Map3D map) {

  final float tileSize = 25.0f;
  this.map = map;

  float w = (float)Map3D.width;
  float h = (float)Map3D.height;

  // Shadow shape
  this.shadow = createShape();
  this.shadow.beginShape(QUADS);
  this.shadow.fill(0x992F2F2F);
  this.shadow.noStroke();
  // INSÉREZ VOTRE CODE ICI
  this.shadow.endShape();

  // Wireframe shape
  this.wireFrame = createShape();
  this.wireFrame.beginShape(QUADS);
  this.wireFrame.noFill();
  this.wireFrame.stroke(#888888);
  this.wireFrame.strokeWeight(0.5f);
  // INSÉREZ VOTRE CODE ICI
  this.wireFrame.endShape();

  // Shapes initial visibility
  this.shadow.setVisible(true);
  this.wireFrame.setVisible(true);
}
```

Une méthode *update* permettra d'afficher les formes de cette classe à l'écran.

Une méthode *toggle* permettra de rendre visibles ou non visibles les formes de cette classe.

Dans le programme principal, vous déclarez une variable globale *Land land* que vous construirez dans la fonction *setup* de Processing en créant un nouvel objet : *this.land = new Land(this.map);*

Enfin, l'affichage au cours de la fonction *draw* sera réalisé par un appel à la méthode de la classe par *this.land.update();*

Vous pouvez maintenant ajouter une nouvelle commande utilisateur, la touche « w » qui permettra d'afficher ou de masquer le terrain à l'écran :

```
void keyPressed() {
  switch (key) {
  …
  case 'w':
  case 'W':
    // Hide/Show Land
    this.land.toggle();
    break;
  …
  }
}
```

Félicitations, vous avez modélisé votre premier terrain en 3D :

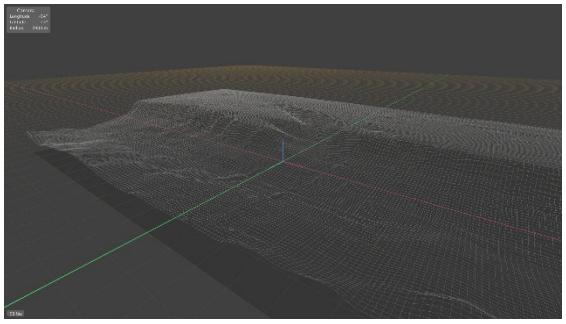

Figure 12 - Maillage en fil de fer du terrain

3.4) Application d'une texture à partir d'image satellite.

Vous allez maintenant appliquer une texture à votre terrain. Pour cela, vous disposez d'une vue satellite « *paris_saclay.jpg* » à placer dans votre dossier « *data* ».

Pour modéliser la vue satellite, vous allez créer dans votre classe *Land* une nouvelle forme *satellite*, à partir d'une copie du code la forme wireframe.

Modifiez le constructeur de la classe *Land* pour recevoir le nom du fichier de texture.

Figure 13 - Vue satellite de Paris Saclay

Vous chargerez le fichier de texture après avoir vérifié sa présence par un test tel que :

```
File ressource = dataFile(fileName);
if (!ressource.exists() || ressource.isDirectory()) {
  println("ERROR: Land texture file " + fileName + " not found.");
  exitActual();
}
PImage uvmap = loadImage(fileName);
```

Vous devez modifier la construction de la forme pour ajouter la texture via une instruction *this.satellite.texture(uvmap);* et indiquer les coordonnées u & v de la texture comme vu en TP.

À la fin du constructeur de classe *Land*, affichez par défaut la forme *satellite* :

```
    // Shapes initial visibility
    this.shadow.setVisible(true);
    this.wireFrame.setVisible(false);
    this.satellite.setVisible(true);
```

Enfin vous modifierez la méthode *toggle* pour permettre à l'utilisateur de basculer entre les formes *wireframe* et *satellite* (*l'ombre portée restera toujours visible*) :

```
/**
 * Toogle between satellite and wireframe mode
 */
void toggle() {
  this.wireFrame.setVisible(!this.wireFrame.isVisible());
  this.satellite.setVisible(!this.satellite.isVisible());
}
```

Félicitations, vous disposez désormais d'une vue satellite de votre terrain en 3D :

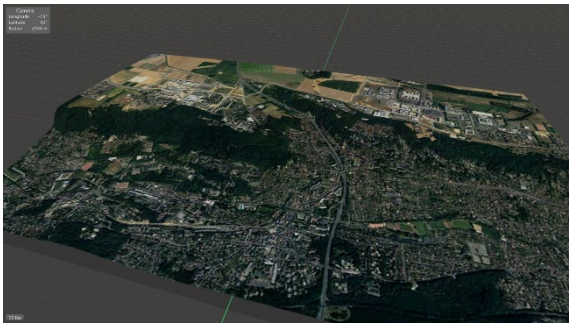

Figure 14 - Texture du terrain à partir d'une image satellite

3.5) Amélioration de l'éclairage.

3.5.1) Calcul des normales.

À ce stade, l'éclairage par défaut donne un résultat acceptable.

Pour gérer l'éclairage à votre guise, vous devez apporter quelques améliorations.

Tout d'abord, vous pouvez régler l'émissivité de la texture avant de créer les points :

```
this.satellite.emissive(0xD0);
```

Ensuite, vous pouvez approximer aisément les normales des points du terrain, à l'aide de la méthode *toNormal* de l'objet *ObjectPoint* juste avant de créer chaque *vertex* :

```
ObjectPoint op = this.map.new ObjectPoint(x, y);
PVector n = op.toNormal();
this.satellite.normal(n.x, n.y, n.z);
this.satellite.vertex(op.x, op.y, op.z, u, v);
```

3.5.2) Éclairage vertical personnalisé.

Vous pouvez maintenant ajouter un éclairage personnalisé, dans la classe *Camera*.

Ajouter la variable booléenne *lightning* à la classe *Camera*, initialisez là à *false* par défaut.

Ajoutez à cette classe une méthode *toggle* qui permet de d'inverser sa valeur.

Enfin, modifiez la méthode *update* de la caméra pour ajouter votre éclairage personnalisé après le positionnement de la caméra, par exemple :

```
// Sunny vertical lightning
ambientLight(0x7F, 0x7F, 0x7F);
if (lightning)
  directionalLight(0xA0, 0xA0, 0x60, 0, 0, -1);
lightFalloff(0.0f, 0.0f, 1.0f);
lightSpecular(0.0f, 0.0f, 0.0f);
```

***Important** : Prenez soin d'afficher les informations utilisateurs à l'écran par this.hud.update(); en toute fin de la fonction draw, car cette méthode annule les réglages d'éclairages.*

Dans le programme principal, modifiez votre gestion du clavier pour pouvoir allumer ou éteindre la lumière directionnelle avec la touche « *L* ».

Félicitations, vous pouvez désormais obtenir une éclaircie ensoleillée à la demande…